UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PROGRAMA DE PÓS-GRADUAÇÃO EM
CIÊNCIA DA COMPUTAÇÃO

MARLON LAMARTINE REGES SILVA

IMPLEMENTAÇÃO DO *Benchmark Linpack* PARA AVALIAÇÃO DE DESEMPENHO DE NODOS SENSORES COM DIFERENTES NÍVEIS DE ENERGIA

MOSSORÓ, RN
2014

Marlon Lamartine Reges Silva

Implementação do Benchmark Linpack para Avaliação de Desempenho de Nodos Sensores com Diferentes Níveis de Energia

Dissertação de Mestrado submetida ao Programa de Pós-graduação em Ciência da Computação da Universidade do Estado do Rio Grande do Norte e Universidade Federal Rural do Semi-Árido como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Pedro Fernandes Ribeiro Neto
Co-Orientador: Prof. Dr. Henrique Jorge de Amorim Holanda

MOSSORÓ, RN
2014

Marlon Lamartine Reges Silva

Implementação do Benchmark Linpack para Avaliação de Desempenho de Nodos Sensores com Diferentes Níveis de Energia

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação para obtenção do grau de Mestre em Ciência da Computação Aprovada em 24 de Fevereiro de 2014

Profº. Dr. Pedro Fernandes Ribeiro Neto
Orientador
Universidade do Estado do Rio Grande do Norte (UERN)

Profº. Dr. Henrique Jorge de Amorim Holanda
Co-Orientador
Universidade do Estado do Rio Grande do Norte (UERN)

Profº. Dr. Silvio Roberto Fernandes de Araújo
Membro Interno
Universidade Federal Rural do Semi-Árido (UFERSA)

Profº. Dr. Ricardo Alexsandro de Medeiros Valentim
Membro Externo
Universidade Federal do Rio Grande do Norte (UFRN)

MOSSORÓ, RN, 2014

Dedicatória

Dedico esta Dissertação a minha família,
minha irmã Daniele, minha mãe Socorro,
e ao meu pai Mariano (in memoriam),
que sempre me apoiaram em tudo.
Muito Obrigado!

Agradecimentos

A conclusão de um objetivo é o momento ideal para reflexão dos acontecimentos, fatos e situações que nos trouxeram até este momento. Toda essa trajetória percorrida, cada etapa superada é uma pequena conquista nas nossas vidas. Tudo isso não seria possível sem o auxílio e acompanhamento das pessoas que estão ao nosso lado diariamente. Por isso, cada uma destas pessoas merece um momento de agradecimento.

Como forma de agradecimento, dedico esta dissertação à Deus, ser divino de graça e bondade, que nos ilumina diariamente com sua luz, nos trazendo paz e fé, para continuarmos seguindo atrás de nossos objetivos.

Agradeço à minha família, minha mãe Socorro, minha irmã Daniele, e a meu pai Mariano (in memoriam), por sempre me apoiarem em cada decisão que eu já tomei, e por me incentivarem e não me deixarem desistir diante das dificuldades.

Agradeço à minha namorada Samara por estar sempre ao meu lado me apoiando e me incentivando, independente de qualquer dificuldade e da falta de tempo para ela.

Agradeço a meu orientador Pedro Fernandes pela orientação e por acreditar no meu potencial, quando poucas pessoas acreditaram.

Agradeço aos meus amigos de laboratório, Marlos, Anderson, Kayo, Nathan, Rodrigo, Irlan, Bruno, Leandro, Davi, Suellem, Jomar, Ciro e a meu primo Ronnison, que por muito tempo foram minha família em Mossoró.

Aos meus amigos que já passaram pelo laboratório e que hoje seguem suas carreiras profissionais, Welliana, Márcia, Cleone, Lenardo, Luana, Karla Haryanna que mesmo não estando mais presente diariamente, tenho certeza que ainda torcem pelas minhas conquistas.

Agradeço aos meus amigos do curso de ciência da computação, Aristóteles (Totim), Ismael, Vladimir, João Carias, Romarim, Kayo César, Vitor (Mario), Fernando.

Aos meus professores do mestrado e da graduação, por me incentivarem a estudar e aprender mais sobre esta área grandiosa que é ciência da computação.

Agradeço à UERN e a UFERSA pela oportunidade de aperfeiçoamento acadêmico e infraestrutura fornecida, bem como a CAPES pelo apoio financeiro.

Por fim, a todos que esqueci de mencionar aqui. Meus mais sinceros agradecimentos.

Epígrafe

Que os vossos esforços desafiem as impossibilidades, lembrai-vos de que as grandes coisas do homem foram conquistadas do que parecia impossível.

CHARLES CHAPLIN

Resumo

Uma rede de sensores sem fio é formada por um conjunto de dispositivos denominados de nodos sensores que são distribuídos em uma área geográfica com o intuito de sensoriar e monitorar fenômenos físicos. Cada nodo sensor possui um conjunto de componentes, onde estes por sua vez, possuem uma funcionalidade diferente. Dependendo da aplicação e/ou do nodo sensor, estes podem ter seu abastecimento de energia limitado, dessa forma, especificar e programar o nodo a fim de gastar o mínimo de energia possível para assim prolongar o tempo de vida do nodo se torna uma tarefa quase primordial. No entanto, esse desgaste de energia pode influenciar no que diz respeito aos componentes do nodo, em especial o microcontrolador. Além disso, muitas pesquisas relacionadas com este tipo de pesquisa, focam apenas no consumo de energia gasto com base no processamento do microcontrolador. No entanto, a abordagem desta pesquisa foca nos valores de processamento com base nas oscilações de energia. Assim, o objetivo desta dissertação é avaliar o desempenho de microcontroladores, tendo como base o controle da quantidade de energia que é fornecida ao nodo sensor, e verificar se as variações de energia influenciam no processamento do microcontrolador. Para isso, foi implementado o modelo de benchmark Linpack em NesC, para auxiliar neste processo de avaliação. Ao fim, é comprovado que a energia do nodo sensor pode influenciar no seu processamento.

Palavras-chave: Nodo Sensor, Avaliação de Desempenho, Benchmark, Linpack.

Abstract

The Wireless Sensor Networks is composed by a set of devices called sensors nodes, which are distributed in a geographical area in order to sensing and monitoring physical phenomena. Each sensor node has a set of components, these which one has one functionality. Depending on the application and/or sensor node, they can have their energy supply limited, thus, specify and program the node in order to spend as little as possible so as to prolong the node's energy life, almost become a essential task. However, this spent of energy may affect of the node's components, in particular the microcontroller. In addition, many researches related with this kind of research, focuses only on energy consumption based on microcontroller's processing. However, the approach of this research focuses on processing values based on the oscillations of energy. Thereby, this dissertation proposes to evaluate the performance of microcontroller, based on controlling the amount of energy that is supplied to the sensor node, and verify if the power variations influence the microcontroller's processing. For this, the Linpack benchmark was implementd in NesC, to assist in this evaluate process. At the end, it is shown that the energy of the sensor node may influence in it's processing.

Keywords: Sensor Node, Performance Evaluation, Benchmark, Linpack.

Lista de Símbolos e Abreviaturas

Abreviatura	**Descrição**
ADC	*Analog-To-Digital Converter*
*BSM*1	*Benchmark Simulation 1*
CMOS	*Complementary Metal-Oxide-Semicondutor*
CPV	*Component Parameter Value*
DMIPS	*Dhrystone Millions Instructions Per Second*
DSP	*Digital Signal Processor*
EEMBC	*Embedded Microprocessor Benchmark Consortium*
EEPROM	*Electricallly-Erasable Programmable Read-Only Memory*
E/*S*	*Entrada/Saída*
FIR	*Finite Impulse Response*
FLOP/*S*	*Floating-point Operations Per Second*
FPGA	*Field Programmable Gate Array*
IGU	*Interface Gráfica de Usuário*
KBPS	*Kilo Bits Per Second*
KWIPS	*Kilo Whetstone Instructions Per Second*
LAN	*Local Area Network*
MBPS	*Mega Bits Per Second*
MFLOP/*S*	*Millions Floating-point Operations Per Second*
MHZ	*Mega Hertz*
MPI	*Message Passing Interface*
OpenMP	*Open Multi-Processing*
OPS	*Operações Por Segundo*
RISC	*Reduced Instruction Set Computer*
RSSF	*Redes de Sensores Sem Fio*
SIMD	*Single Instruction Multiple Data*
SIP	*Session Initiation Protocol*
SMP	*Symmetric MultiProcessing*
SOA	*Service Oriented Architecture*
SPEC	*Standard Performance Evaluation Corporation*
SRAM	*Static Random Access Memory*
USB	*Universal Serial Bus*

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

1.1 Contextualização

Uma rede de sensores é composta por um conjunto de dispositivos denominados de nodos sensores, distribuídos em uma área geográfica para monitoramentos de fênomenos físicos (ANASTASI et al., 2009). Os avanços nas pesquisas desta tecnologia têm permitido a sua utilização para os mais diversos fins. Missões espaciais não tripuladas, explorações subaquáticas e interplanetárias são exemplos de ambientes críticos e perigosos que podem se beneficiar com a utilização de redes de sensores sem fio (Borges Neto, 2009).

Nessas redes, cada nodo é equipado com uma variedade de sensores, tais como acústico, sísmico, infravermelho, vídeo-câmera, calor, temperatura, pressão e umidade. Esses nodos podem ser organizados em grupos (*clusters*) onde pelo menos um dos sensores deve ser capaz de detectar um evento na região, processá-lo e tomar uma decisão se deve fazer ou não uma difusão (*broadcast*) do resultado para outros nodos. O objetivo é que redes de sensores sem fio se tornem disponíveis em todos os lugares executando as tarefas mais diferentes possíveis (LOUREIRO et al., 2003).

Embora estes dispositivos possuam diversas vantagens com a sua utilização, por outro lado também apresentam limitações. Estas limitações existentes para os nodos sensores são fatores que impulsionam a necessidade de diversas outras adaptações no hardware destes dispositivos.

Como herança dos sistemas de computação embarcada, que formam a base da arquitetura de um sensor, a necessidade de produzir sensores mais baratos impulsionou as pesquisas a direcionar o foco na redução das estruturas destes dispositivos, sem que isto afete na eficiência do nodo

sensor (Borges Neto, 2009).

Uma vez que os sensores possuem tamanhos cada vez mais reduzidos, a capacidade de alimentação de energia também foi reduzida (Borges Neto, 2009). Com isso, outros aspectos como o poder de processamento, espaço de armazenamento e potência de transmissão também foram reduzidos. Assim, todos os componentes do sensor precisam compartilhar da mesma fonte de alimentação, reduzida e limitada.

Como a capacidade de energia do nodo sensor é limitada, a eficiência desta capacidade se torna um importante requisito a ser atendido em um nodo sensor sem fio. Estes requisitos para baixo consumo de energia e tempo de vida mais prolongado da rede podem ser alcançados por intermédio de compromissos entre desempenho computacional e consumo de energia.

Dentre os componentes do nodo sensor, a subcamada de processamento compreende a memória e um microcontrolador para processamento de dados locais. Segundo Anastasi et al. (2009), medições experimentais têm mostrado que geralmente transmissões de dados são muito mais custosas em termos de consumo de energia, enquanto processamento de dados consome significativamente menos.

Contudo, redução de energia em microcontroladores é complicado, devido ao fato que é exigido que os microcontroladores possuam poder computacional crescente, flexibilidade e consumo de energia muito baixo (PESOVIC et al., 2012). Estas características são contraditórias por si só, pois projetar um sistema com consumo de energia reduzido sem que isto afete seu desempenho pode ser uma tarefa árdua.

Sendo assim, para avaliar o desempenho de microcontroladores e poder avaliar o consumo de energia do nodo, utiliza-se *benchmark*, uma técnica bastante utilizada para avaliar o desempenho de computadores. *Benchmarks* utilizam cargas de trabalho para simular situações de processamento em que um microcontrolador pode ser submetido. Embora, a maioria dos *benchmarks* encontrados em pesquisas possuam métricas voltadas para processadores específicos. O ideal é utilizar um modelo que possua características adequadas para avaliação de processamento e do tempo levado para realizar tal processamento.

Existem diversos modelos de *benchmarks* disponíveis, como *SPEC* (SPEC, 2014), que possui foco na avaliação de diversas tecnologias, a fim de determinar um padrão de *benchmark* e de avaliação para cada tecnologia. O *benchmark Dhrystone* é um modelo sintético que possui base em cálculos de inteiros, já o modelo *Whetstone* foca no cálculo de pontos-flutuantes. O modelo

Linpack produz uma carga de trabalho com base na resolução de sistemas de equações lineares, tendo como métricas de desempenho o tempo de processamento e o cálculo de MFLOP/S.

Estes modelos de *benchmarks* foram desenvolvidos inicialmente para computadores, embora possam ser utilizados para avaliar outros sistemas computacionais, inclusive redes de sensores sem fio. Com relação a nodos sensores, a avaliação nestes geralmente focam no desgaste de energia do microcontrolador, no tempo de processamento e o cálculo de operações por segundo.

Por fim, com uma observação relacionada com as variações de energia decorrentes das atividades do nodo, além da influência do ambiente, podem acarretar em leituras de processamento errôneas e influenciar no tempo destas. Dessa forma, uma análise de leituras de processamento com níveis de energia variáveis durante este processamento, torna-se necessária.

1.2 Motivação

Quando se trata de desempenho de microcontroladores para avaliar o consumo de energia do nodo, inicialmente o foco se concentra na redução do gasto de energia do microcontrolador.

Uma técnica para redução de energia do nodo é alternar entre funcionalidade e energia adaptando a velocidade com que o controlador opera. A idéia é escolher a melhor velocidade possível para processar uma tarefa que tem de ser completada dentro de um tempo limite. A solução foca em alternar o controlador para o modo de operação completa, processar a tarefa em alta velocidade e voltar para o modo *sleep* (KARL; WILLIG, 2005). Esta técnica abordada em um nodo que seja requisitado a processar em intervalos de tempos pequenos pode vir a ser desgastante.

Contudo, a maioria das pesquisas focam na avaliação e comparação de um conjunto de processadores ou de um processador em específico. Uma destas, realizada por Nazhandali, Minuth e Austin (2005), em que o foco do trabalho é desenvolver um conjunto de aplicações para representar cargas de trabalho de tempo-real, além de novas métricas de desempenho adequadas para avaliar microprocessadores de sensores sem fio.

Já as pesquisas realizadas por Hempstead, Welsh e Brooks (2004), possui como foco o desenvolvimento de um *benchmark* padronizado para avaliação de hardware de nodos sensores. No entanto, essa avaliação não possui foco apenas no processador nem menciona níveis de energia, além disso a forma de avaliação deste é por intermédio de estresses do processamento.

Esta dissertação toma como base as pesquisas realizadas por Pesovic et al. (2012), onde fo-

ram avaliados modelos de processadores diferentes e utilizando modelos de *benchmark* diferentes, para observar qual modelo de processador possui melhor rendimento com base no tempo de execução e no tamanho de código gerado por cada *benchmark*. Além disso, foram avaliados os modos de operação ativo e *sleep* de cada processador. No entanto, este analisa o consumo de energia de cada processador em modo ativo, mas não analisa os resultados do processamento com variações deste consumo.

Tendo estas informações como motivação, esta dissertação apresenta uma avaliação de desempenho de nodos sensores sem fio, utilizando o *benchmark Linpack*, para análise dos resultados de processamento com variações de energia.

1.3 Objetivo Geral

O objetivo desta dissertação é implementar o *benchmark Linpack* em linguagem de programação NesC para avaliar o desempenho de nodos sensores sem fio, com base em variações de energia durante o processamento da carga de trabalho e observar os resultados deste processamento.

1.4 Objetivos específicos

Para alcançar os objetivos deste trabalho, as seguintes atividades específicas devem ser executadas:

1. Definição das diretrizes do processo de avaliação;
2. Implementação do *benchmark Linpack*, adequando suas características de execução para nodos sensores sem fio;
3. Avaliação dos nodos sensores utilizando o *benchmark* implementado;
4. Análise dos resultados obtidos;
5. Comparação dos resultados dos nodos sensores.

1.5 Organização da Dissertação

Esta dissertação está organizada em capítulos que por sua vez estão organizados em sessões. No Capítulo 2 são apresentadas as definições de redes sensores sem fio e do nodo sensor, assim como

as subcamadas da sua arquitetura, em especial a subcamada de processamento. No Capítulo 3, são explicadas as definições, categorização e distinção de métricas relacionados com o processo de avaliação de desempenho. O Capítulo 4, apresenta a definição de carga de trabalho, o processo de modelagem de carga de trabalho, além de alguns modelos de *benchmarks* conhecidos na literatura. O Capítulo 5, apresenta a implementação do *benchmark Linpack*, além do processo de avaliação e análise dos resultados obtidos. No Capítulo 6, são realizadas as considerações finais referentes a pesquisa, bem como sua contribuição e trabalhos futuros.

Capítulo 2

Nodo Sensor

Redes de Sensores Sem Fio é uma tecnologia emergente e que possui diversas áreas de aplicações, além disso têm sido o foco de inúmeras pesquisas científicas nos últimos anos, que vão desde o melhor aproveitamento dos recursos dos nodos sensores, até sua implantação em ambientes hostis e inacessíveis para o homem.

Neste capítulo serão apresentadas as definições acerca de redes de sensores sem fio, mas com foco no nodo sensor. Na Seção 2.1 são introduzidos conceitos e aplicações envolvendo redes de sensores sem fio. Na Seção 2.2 é explicada a arquitetura de um nodo sensor e suas subcamadas, com ênfase na subcamada de processamento. A Seção 2.3 resume algumas características acerca do consumo de energia do nodo sensor e também da subcamada de processamento. Por fim, a Seção 2.4 apresenta as considerações finais.

2.1 Redes de Sensores Sem Fio

O avanço que tem ocorrido na área de micro-processadores, novos materiais de sensoriamento, micro sistemas eletromecânicos e comunicação sem fio, tem estimulado o desenvolvimento e uso de sensores inteligentes em áreas ligadas a processos físicos, químicos, biológicos, dentre outros. Com a união destas tecnologias, surgiu um novo paradigma de comunicação, conhecido como Redes de Sensores Sem Fio (RSSF) (LOUREIRO et al., 2003) (Borges Neto, 2009).

Estes sensores interligam o ambiente físico com o mundo digital, capturando e revelando fenômenos e convertendo-os em uma forma que pode ser processada, armazenada e atuada. Integrados em grande quantidade de dispositivos, máquinas e ambientes, os sensores fornecem benefícios

sociais. Como por exemplo, evitar falhas catastróficas em infraestruturas, conservar recursos naturais preciosos, aumentar produtividade, reforçar a segurança, além de possibilitar novas aplicações tais como, sistemas sensíveis ao contexto e tecnologias *smart home* (DARGIE; POELLABAUER, 2010).

2.1.1 Aplicações

Redes de sensores sem fio possui muitas aplicações. Algumas destas são futurísticas, enquanto uma grande quantidade são praticamente utilizáveis, como monitoramento de ambientes, rastreamento de objetos e/ou animais, monitoramento de condutores (água, óleo, gás), dentre outras. Dargie e Poellabauer (2010) lista alguns cenários e possíveis aplicações:

Monitoramento de estruturas: em 2007 alguns desastres envolvendo desabamentos de pontes e suas estruturas arruínadas ou comprometidas, incentivaram a implantação de nós sensores para inspeções de estruturas físicas. Estes nós são alocados em áreas inacessíveis para dispositivos volumosos e cabeados. Além disso, empregando uma grande quantidade de nós, é possível estabelecer uma correlação entre diferentes medições, o que facilita a localização de danos. Por fim, o desenvolvimento e a manutenção da rede de sensores não necessita de paralisias na operação normal da estrutura.

Controle de tráfego: transporte terrestre é uma infra-estrutra complexa e vital, pois fornece suporte a uma variedade de sistemas como, cadeia de suprimentos e saúde pública. Em áreas urbanas isto resulta em congestionamento em potencial. Um relatório de mobilidade urbana, emitido pelo *Texas Transportation Institute*, revelou que em 2007 o congestionamento causado pelos americanos, obteve um custo estimado em aproximadamente $87,2$ bilhões de dólares. Uma abordagem para lidar com congestionamentos seria por sistemas de sensoriamento que reduzem o congestionamento. Esses sistemas coletam informações sobre a densidade e velocidades dos veículos nas estradas, e sugere aos condutores algumas rotas alternativas e saídas de emergências.

Cuidados médicos: um grande leque de aplicações de cuidados de saúde têm sido propostos para redes de sensores sem fio, incluindo monitoramento de pacientes com doenças de Parkinson, epilepsia, pacientes com doenças do coração, pacientes em reabilitação de ataques cardíacos e pessoas idosas. Enquanto cuidados de saúde preventiva têm sido defendido por muitos como o principal meio de reduzir os gastos de saúde e taxas de mortalidade, estudos mostram que para alguns pacientes certas práticas são inconvenientes, complicadas e interferem com suas vidas

diárias. Como perdas de visitas de *checkup* ou sessões de terapia devido a choque de horários, ou mesmo custo de transporte. Um exemplo de aplicação de RSSF nessa área seria desenvolver sistemas de monitoramento de saúde confiável e discreto, que auxiliam os pacientes a reduzir as responsabilidades e presença pessoal de médicos, além de alertar estes e enfermeiras quando da necessidade de intervenção médica necessária.

Para todos estes cenários o pleno funcionamento dos sensores é útil tanto para o consumo de energia dos nodos como para a própria rede, afetando diretamente no seu desempenho. Dessa forma, definir um balanceamento entre energia e desempenho para um conjunto de nodos torna-se essencial para qualquer aplicação.

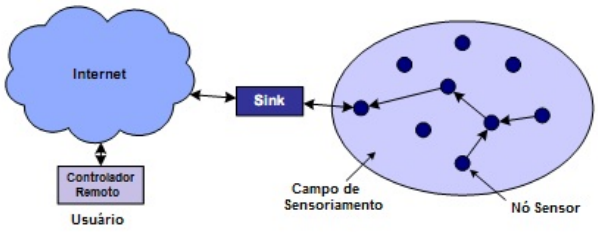

Figura 2.1: Arquitetura de uma rede de sensores. Adaptado de (ANASTASI et al., 2009)

A Figura 2.1 representa uma arquitetura de uma rede de sensores em que consiste um nodo *sink* (ou *estação base*), e uma grande quantidade de nodos sensores implantados sobre uma grande área geográfica (*campo de sensoriamento*). As informações sensoriadas são convertidas para dados e depois transferidos a partir dos nodos sensores para o nó sink por intermédio de um paradigma de comunicação multi-salto, onde os dados são enviados de um nodo para outro até chegar no seu destino que é o nodo sink.

A partir deste modelo pode-se perceber a importância de cada nodo na rede, pois o funcionamento destes nodos possibilitam à rede a execução das suas atividades. Por outro lado, se algum destes nodos forem desligados, removidos ou ter sua energia finalizada, pode comprometer o funcionamento de uma região ou mesmo da rede como um todo.

Para Dargie e Poellabauer (2010), os nodos sensores são o elemento central em uma rede de sensores. É por intermédio destes que são realizados os processos de sensoriamento, processamento e comunicação. Os nodos armazenam e executam os protocolos de comunicação e os algoritmos de processamento de dados. No entanto, a qualidade, tamanho e a frequência dos dados sensoriados que podem ser extraídos a partir da rede, são influenciados pelos recursos físicos

disponíveis no nodo. Portanto, o projeto e implementação de um nodo sensor sem fio e dos seus componentes é uma fase importante da sua implementação.

2.2 Arquitetura de um Nodo Sensor

Um nodo sensor é formado basicamente por subcamadas e pelos componentes que as representam, além de componentes externos que podem ser necessários por uma aplicação específica.

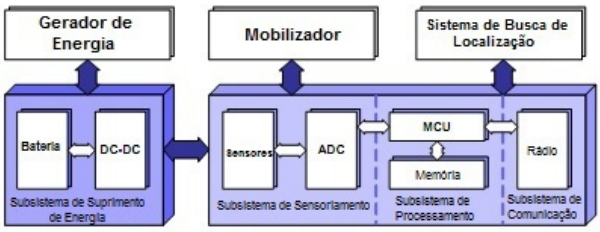

Figura 2.2: Arquitetura de um nodo sensor. Adaptado de (ANASTASI et al., 2009)

A Figura 2.2 apresenta uma arquitetura de um nodo sensor, onde são exibidas suas subcamadas, separadas da unidade de suprimento de energia, além de possíveis componentes externos e onde estes se conectam.

Para Anastasi et al. (2009) um nodo sensor possui quatro componentes principais:

- o *subsistema de sensoriamento* inclui um ou mais sensores (com conversores de analógicos para digitais) para aquisições dos dados do ambiente;
- o *susbsistema de processamento* inclui um microcontrolador e uma memória para processamento local de dados;
- o *subsistema de comunicação* para comunicação de dados *wireless*;
- *unidade de suprimento de energia* para suprir a carga necessária pelo dispositivo para desempenhar as tarefas programadas;

Dependendo da aplicação específica, os sensores podem também incluir componentes adicionais tais como *sistema de busca de localização* para determinar suas posições, ou mesmo um *mobilizador* para mudar suas localizações ou configurações (por exemplo, orientação da antena).

Cada subcamada possui suas características além da sua vital importância no funcionamento do nodo sensor e sua influência no gasto de energia do dispositivo. A seguir serão explicados

alguns detalhes relacionados a estas subcamadas e suas influências no plano de energia do nodo sensor, em especial a subcamada de processamento que é o foco desta pesquisa.

2.2.1 Subcamada de Sensoriamento

Sensoriar fenômenos físicos não é uma novidade, no entanto, o advento de sistemas microeletromecânicos têm tornado o sensoriamento um processo ubíquo. Atualmente há uma infinidade de sensores que medem e quantificam atributos físicos de forma econômica. Segundo Dargie e Poellabauer (2010), os sensores realizam a interface entre o mundo virtual com o mundo físico. A importância do subsistema de sensoriamento é integrar um ou mais sensores físicos e fornecer um ou mais conversores analógicos-digitais bem como um mecanismo de multiplexação para compartilhá-los.

Um sensor físico contém um transdutor, um dispositivo que converte uma forma de energia em outra forma de energia. A saída desse transdutor é um sinal analógico que possui magnitude contínua assim como uma fonte de tempo. Portanto, um conversor analógico para digital é necessário para realizar uma interface entre o subsistema de sensoriamento e o processador digital (DARGIE; POELLABAUER, 2010).

O consumo de energia da subcamada de sensoriamento depende especificamente do tipo do sensor. Em muitos casos a energia consumida torna-se desprezível em relação à energia consumida pelo processamento, e acima de tudo, pela subcamada de comunicação. Em outros casos, o gasto de energia para o sensoriamento de dados pode ser comparável, ou mesmo superior, do que a energia necessária para transmissão de dados. No geral, as técnicas de economia de energia focam em duas subcamadas, subcamada de comunicação (o gerenciamento de energia leva em consideração cada operação de cada nodo sensor por si, bem como na concepção dos protocolos de rede), e a subcamada de sensoriamento (as técnicas são usadas para reduzir a quantidade ou frequência de amostras de energia custosas) (ANASTASI et al., 2009).

2.2.2 Subcamada de Comunicação

Assim como a escolha do modelo adequado de processador é vital para o desempenho, bem como o consumo de energia de um nodo sensor, o modo como os componentes estão interconectados com o subsistema de processamento também é vital.

A transferência de dados eficiente e rápida entre os subsistemas de um nodo sensor é crítica para a eficiência da rede como um todo. No entanto, o tamanho do nodo põe uma restrição nos barramentos do sistema. Considerando que a comunicação via barramento paralelo é mais rápida do que via barramento serial, um barramento paralelo necessita de mais espaço. Além do mais, este necessita de uma linha dedicada para cada bit que deve ser transmitido simultâneamente enquanto o barramento serial necessita somente de uma única linha de dados simples (DARGIE; POELLABAUER, 2010).

2.2.3 Suprimento de Energia

O tempo de vida de um sensor depende da quantidade de energia disponível, e essa fonte de energia geralmente consiste de uma bateria com capacidade de energia limitada. Aplicações, protocolos e algoritmos para RSSFs não podem ser escolhidos por sua elegância e capacidade, mas definitivamente pela quantidade de energia consumida.

Além do mais, pode ser inconveniente, e em alguns casos impossível, recarregar ou trocar esta bateria. Isto se deve ao fato de que alguns nodos sensores podem ser implantados em ambientes hostis ou inaptos. Por outro lado, uma rede de sensores devem ter um tempo de vida longo o suficiente para cumprir as restrições da aplicação (ANASTASI et al., 2009) (LOUREIRO et al., 2003).

De qualquer forma, energia é um recurso crítico e deve ser utilizado moderadamente. Portanto, conservação de energia é uma característica chave em um modelo de sistema como uma rede de sensores sem fio. Principalmente se considerado os componentes dos subsistemas desta rede.

2.2.4 Subcamada de Processamento

O subsistema de processamento reúne todos os outros subsistemas e alguns periféricos adicionais. Seu principal propósito é processar (executar) instruções relativas ao sensoriamento, comunicação e auto-organização. Este subsistema consiste de um chip processador, uma memória não-volátil (geralmente uma memória flash interna) para armazenar instruções de programa, uma memória ativa para armazenar temporariamente os dados sensoriados, e um *clock* interno (DARGIE; POELLABAUER, 2010).

Considerando que existam uma grande quantidade de processadores disponíveis no mercado para utilizar em um nodo sensor sem fio, torna-se necessário realizar uma escolha cuidadosa, uma vez que esta afeta diretamente o custo, flexibilidade, desempenho e consumo de energia

do nodo. Se a tarefa de sensoriamento é bem definida desde o princípio, e não se modifica ao longo do tempo, um projetista pode escolher um FPGA ou um processador de sinal digital. Esses processadores são muito eficientes em termos do seu consumo de energia, e para as tarefas de sensoriamento mais simples, são bem adequados. No entanto, como estes não são os processadores de uso geral, o processo de concepção e implementação pode ser complexo e dispendioso.

Uma solução para este fato é utilizar processadores de propósito específico, assim como computadores *desktop*. Esses processadores são altamente sobrecarregados e seus consumos de energia são excessivos. No entanto, os processadores mais simples que existem, possuem seu uso voltado especificamente para sistemas embarcados. Estes processadores são comumente referenciados como microcontroladores.

Para Dargie e Poellabauer (2010), a maioria dos nodos sensores utilizam microcontroladores devido ao fato que RSSFs é uma tecnologia emergente, e a comunidade acadêmica está ativa com relação ao desenvolvimento de protocolos de comunicação eficientes de energia e algoritmos de processamento de sinais. Como estes necessitam de instalação e atualização de códigos dinâmicos, um microcontrolador é a melhor opção.

O microcontrolador é o núcleo de um nó sensor sem fio. É definido como um computador em um circuito integrado simples, consistindo de uma unidade central de processamento relativamente simples e também por componentes adicionais, tais como barramentos de alta velocidade, unidade de memória, temporizador de vigilância e um clock externo. Este coleta dados dos sensores, processa tais dados, decide quando e para onde enviá-lo, recebe dados de outros nodos sensores, e decide sobre o comportamento do atuador. Tem de executar vários programas, que vão desde o processamento de sinais em tempo crítico e protocolos de comunicação até programas de aplicação (KARL; WILLIG, 2005) (DARGIE; POELLABAUER, 2010).

As características-chave que tornam os microcontroladores particularmente adequados para sistemas embarcados são, sua flexibilidade em conectar com outros dispositivos (como sensores), seu conjunto de instruções propícios ao processamento de sinais de tempo crítico, são livremente programáveis e portanto mais flexíveis, e geralmente possuem memória já embutida. Além disso, sua construção compacta, tamanho pequeno, baixo consumo de energia, e baixo custo tornam este adequado para construção de aplicações autônomas computacionalmente menos intensivas. Microcontroladores são também adequados para RSSFs uma vez que estes comumente têm a possibilidade de reduzir seus consumos de energia entrando em estado de dormência, onde somente

algumas partes do controlador estão ativas.

Outra vantagem que um microcontrolador possui é a flexibilidade de programação. A maioria dos microcontroladores disponíveis comercialmente podem ser programados com linguagem de programação Assembly e C. O uso de linguagens de programação de alto nível aumenta a velocidade de programação e facilita a depuração. Há ambientes de desenvolvimento que oferecem uma abstração de todas as funcionalidades do microcontrolador. Isso permite que desenvolvedores de aplicações para microcontroladores programem sem a necessidade de possuir conhecimento em baixo-nível de hardware.

No entanto, microcontroladores não são tão poderosos e tão eficientes quanto alguns processadores feitos sob medida, tais como DSPs e FPGAs. Além disso, para aplicações que demandam tarefas de sensoriamento simples, ao invés de implementações em larga escala (como em uma agricultura de precisão e monitoramento de vulcões ativos), pode-se preferir usar arquiteturas simples ao invés de processadores eficientes de custo e energia, tais como circuitos integrados específicos de aplicação (KARL; WILLIG, 2005) (DARGIE; POELLABAUER, 2010).

Dentre os microcontroladores que são comumente utilizados em protótipos de redes de sensores sem fio incluem, o MSP 430 da Texas Instruments ou a família de microcontroladores da Atmel, já entre os protótipos mais antigos, têm os processadores StrongArm da Intel, que não são mais frequentemente utilizados por não serem considerados como uma opção prática (KARL; WILLIG, 2005). O microcontrolador utilizado nesta pesquisa foi o ATmega128L, portanto é o microcontrolador que será mais discutido em alguns trechos do documento.

2.3 Consumo de Energia do Nodo Sensor

Muitos trabalhos e pesquisas em RSSFs têm focado na otimização do consumo de energia por intermédio da implementação de protocolos de economia de energia em diferentes camadas do modelo de RSSFs. Os problemas relacionados ao consumo de energia são relatados quando a eficiência da rede diminui depois da perda de alguns sensores devido a uma autonomia de baixo nível da bateria (CHARFI et al., 2010). A rede sem fio se torna ineficiente e o ambiente não pode ser monitorado assim como a rede foi inicialmente projetada. De fato, um dos objetivos em redes sensores é melhorar a função de controle de consumo de energia nos nodos da rede e da estação base para garantir uma vida útil mais longa.

O consumo de energia em um nodo sensor pode ser a partir tanto de fontes úteis quanto dispendiosas. Consumo de energia útil pode ser caracterizado como transmissão e recepção de dados, processamento de solicitações de consultas, e encaminhamento de consultas e dados para nodos vizinhos. O consumo de energia dispendioso pode ser caracterizado como um ou mais dos seguintes fatos. Uma das maiores fontes de desperdício de energia é escuta ociosa (*idle listening*), isto é, escutar a partir de um canal ocioso a fim de receber um possível tráfego. Os outros fatores que contribuem para o desperdício de energia são, colisão, *overhearing* (quando um nodo recebe dados que não são para ele), além de *overhead* de controle de pacotes (REZAEI; MOBININEJAD, 2012).

Atividades dispendiosas ou desnecessárias podem ser classificadas como locais (limitadas ao nodo sensor) ou globais (que possuem escopo voltado para a rede de sensores.) Em ambos os casos, estas atividades podem ser consideradas como acidentes ou resultados não otimizados de hardware e/ou software. Exemplos destas situações em atividade local são; observações acerca de um campo de observação revelam que alguns nodos esgotaram suas baterias prematuramente devido a escutas inesperadas de tráfego que causaram a subcamada de comunicação ficar mais tempo operacional do que o esperado. Similarmente, alguns nodos esgotaram suas baterias prematuramente devido que estes tentaram à toa, estabelecer comunicação com uma rede que não era mais acessível a estes, é um exemplo de atividade global.

Uma estratégia que garante que a energia seja consumida ecnonomicamente é utilizar a estratégia de gerenciamento de energia dinâmica. A estratégia pode ter um escopo local ou global. No escopo local foca em minimizar o consumo de energia de nodos individuais fornecendo a cada subcamada a quantidade de energia que é suficiente para realizar a tarefa designada. Já a estratégia global tenta minimizar o consumo de energia da rede como um todo, definindo estados de sonolência para a rede (DARGIE; POELLABAUER, 2010).

No entanto, considerando apenas as atividades locais, segundo Karl e Willig (2005) os principais consumidores de energia são os controladores, o rádio, a memória, e dependendo do tipo, os sensores.

Controladores embarcados geralmente implementam o conceito de estados operacionais múltiplos, além de ser levemente fácil de controlar. A maioria das subcamadas de processamento existentes utilizam os microcontroladores da Atmel e StrongARM da Intel (DARGIE; POELLABAUER, 2010). Esses microcontroladores podem ser configurados para operar em vários modos

de energia.

No exemplo dado por (DARGIE; POELLABAUER, 2010), temos o microcontrolador ATmega128L que possui seis modos de energia diferentes: ocioso, redução de ruído ADC, economia de energia, desligamento, espera e espera prolongada.

- o modo ocioso interrompe a CPU enquanto permite SRAM, contadores/temporizadores, porta SPI e sistema de interrupção continuem funcionando;
- o modo de desligamento salva o conteúdo dos registros enquanto congela o oscilador e desabilita todas as outras funções do chip até a próxima interrupção ou redefinição de hardware;
- no modo de economia de energia, o temporizador assíncrono continua a executar, permitindo ao usuário manter a base do temporizador enquanto os componentes remanescentes do dispositivo entram em modo de sonolência;
- o modo de redução de ruído ADC para a CPU e todos os módulos de entrada e saída, exceto o temporizador assíncrono e o ADC. O objetivo é minimizar ruídos durante as conversões ADC;
- em modo de espera, um oscilador de cristal/ressonância é executado enquanto os componentes de hardware remanescentes entram em modo de sonolência. Isto permite que iniciação muito rápida combinada com consumo de energia baixo;
- no modo de espera prolongado, ambos o oscilador principal e o temporizador assíncrono continuam a operar.

Além destas configurações, a subcamada de processamento pode operar com diferentes tensões e frequências de clock.

2.4 Conclusão

Redes de sensores sem fio possuem muitas áreas de aplicações e por sua vez modelos de aplicações, todavia, é uma tecnologia que possui restrições e limitações. Dentre estes problemas o mais evidenciado pelos autores, é a restrição de energia. No entanto, mesmo em baixas condições de energia é interessante frisar a importância do funcionamento pleno do nodo sensor, de forma que continue a executar suas tarefas da forma como foram previamente designados.

Uma das maneiras de garantir este tipo de funcionamento dos nodos, é por meio de avaliações e estudos. Avaliar previamente a rede ou os nodos previne o desgaste de energia ou o mal funcionamento dos nodos e possivelmente da rede.

Capítulo 3

Avaliação de Desempenho

O desempenho do hardware é responsável direto pela eficácia do sistema, que inclui não só o hardware mas também o software. Devido a isso, a avaliação de desempenho de um sistema constitui um desafio devido à diversidade e complexidade dos sistemas modernos.

Neste capítulo serão apresentadas as definições e características sobre avaliação de desempenho. Na Seção 3.1 são apresentadas as definições que permeiam o processo de avaliação de desempenho. A Seção 3.2 apresenta uma categorização das técnicas de análises de desempenho. A Seção 3.3 apresenta algumas métricas relevantes para alguns processos de avaliação e em especial alguns conceitos relacionados às métricas em sistemas embarcados. Por fim, a Seção 3.4 apresenta a conclusão.

3.1 Definição

O uso de ferramentas de software é uma das principais técnicas para avaliar e melhorar desempenho de sistemas computacionais. No entanto, a inerente complexidade dos sistemas avaliados e as inovações do campo computacional, dificultam o surgimento de novas técnicas e a melhoria das já existentes (CASALE; GRIBAUDO; SERAZZI, 2011).

Para Boudec (2010), "avaliação de desempenho consiste na quantificação dos serviços disponibilizados por um computador ou sistema de comunicação", ou no caso desta dissertação, por um nodo sensor.

Para Jain (2008), os usuários, projetistas e administradores de sistemas computacionais são interessados em avaliações de desempenho, uma vez que o objetivo destes é obter ou fornecer

o mais alto desempenho com o mais baixo custo. Este objetivo tem resultado em uma evolução contínua de sistemas com desempenho elevado e custos baixos, o que conduz à proliferação atual de estações de trabalho e computadores pessoais, muitos dos quais são bem melhores do que os supercomputadores das gerações anteriores. À medida que a indústria de computadores se torna mais competitiva, torna-se mais importante garantir que a alternativa selecionada forneça o melhor *trade-off* entre custo e desempenho.

Embora seja uma atividade custosa considerando o tempo e os valores associados, avaliação de desempenho é uma ferramenta necessária em praticamente todas as fases do ciclo de vida de um sistema computacional, incluindo sua concepção, fabricação, venda/compra, uso e atualização. Assim, várias decisões de projeto são feitas antes que qualquer prototipagem seja feita. Tomando como exemplo o processo de desenvolvimento de microcontroladores, nos estágios iniciais, quando o projeto está sendo concebido, a avaliação de desempenho é usada para fazer *trade-offs* antecipados do projeto. Normalmente, isso é feito por intermédio de modelos de simulação, uma vez que o projeto é finalizado e está sendo implementado, a simulação é utilizada para avaliar a funcionalidade e desempenho dos subsistemas. Posteriormente, medição de desempenho é realizada depois que o produto está disponível, a fim de compreender o desempenho do sistema real para várias cargas de trabalho do mundo real e identificar modificações para incorporar em projetos futuros (KURIAN, 2002) (JAIN, 2008).

Uma avaliação dessa forma é necessária quando um projetista de um sistema computacional quer comparar uma série de projetos alternativos e encontrar o melhor projeto ou comparar um leque de sistemas e decidir qual é o melhor dentre estes que seja mais adequado para um dado conjunto de aplicações. Avaliações de desempenho de sistemas reais, ajudam a determinar o quão bom este sistema está desempenhando e se há quaisquer melhorias necessárias a serem feitas.

No entanto, a existência de muitas aplicações computacionais, dificulta o estabelecimento de uma medida padrão de desempenho, um ambiente de medição padronizado, ou uma técnica padrão para todos os casos. Dessa forma, o primeiro passo em avaliação de desempenho é selecionar as medidas adequadas de desempenho, o ambiente de medição adequado e as técnicas mais adequadas para uma a dada aplicação.

No entanto, em se tratando de dispositivos embarcados, em especial microcontroladores e/ou microprocessadores, as atividades de projeto e avaliação é um grande desafio, especialmente considerando o fato que um segundo de execução de programa nestes processadores envolvem vários

bilhões de instruções e analisar um segundo de instrução pode envolver lidar com dezenas de bilhões de pedaços de informações.

Para Kurian (2002), no geral, um projeto de microprocessadores e sistemas computacionais envolve vários passos:

- compreender as aplicações e as cargas de trabalho que os sistemas estarão executando;
- procurar projetos inovadores em potenciais ;
- avaliar o desempenho dos projetos candidatos;
- selecionar o melhor projeto.

No entanto, há um certo padrão de etapas que pode ser utilizado em um processo de avaliação de desempenho, que é selecionar a técnica de avaliação adequada para aquele dado sistema, selecionar as métricas cabíveis para alcançar os resultados desejados, e por fim, utilizar as cargas de trabalho apropriadas para realizar os testes necessários naquele sitema em avaliação.

3.2 Categorias de Técnicas de Avaliação de Desempenho

Uma das etapas em um processo de avaliação, é escolher qual a técnica é a mais adequada para aquele sistema em avaliação, sendo categorizadas com base nas estratégias e ferramentas que são utilizadas. Assim, um processo de avaliação de desempenho pode ser classificado em modelagem de desempenho e medição de desempenho. Embora as categorias principais sejam fixas, na tabela a seguir são citadas apenas algumas técnicas e ferramentas que se encaixam nestas categorias.

Tabela 3.1: Classificação de Técnicas de Avaliação de Desempenho. Adaptado de (KURIAN, 2002)

Medição de Desempenho	Contadores de Monitoramento de Desempenho de Microprocessador em Chip	
	Monitoramento de Hardware Off-Chip	
	Monitoramento de Software	
	Instrumentação Microcodificada	
Modelagem de Desempenho	Simulação	Simulação Orientada a Rastros
		Simulação Orientada a Execução
		Simulação de Sistema Completo
		Simulação Orientada a Eventos
		Perfil de Software
	Modelagem Analítica	Modelos Probabilísticos
		Modelos em Fila
		Modelos de Markov
		Modelagem em Rede de Petri

A Tabela 3.1 apresenta uma classificação das técnicas de avaliação de desempenho, apresentando duas categorias principais. A primeira é Medição de Desempenho que é possível somente se o sistema de interesse está disponível para medição e somente se este tem acesso aos parâmetros de interesse, e se divide em mais quatro classificações, Contadores de Monitoramento de Desempenho de Microprocessador em Chip, Monitoramento de Hardware Off-Chip, Monitoramento de Software e Instrumentação Microcodificada.

Já a segunda categoria principal, conhecida como Modelagem de Desempenho é tipicamente usada quando sistemas reais não estão disponíveis para medição ou se os sistemas reais não têm pontos de teste para medir cada detalhe de interesse. Modelagem de Desempenho por sua vez também se divide em duas partes, a primeira é Simulação, que enquadra Simulação Orientada a Rastros, Simulação Orientada a Execução, Simulação de Sistema Completo, Simulação Orientada a Eventos, Perfil de Software. A segunda parte, que é Modelagem Analítica, enquadra Modelos Probabilísticos, Modelos em Fila, Modelos de Markov e Modelagem em Rede de Petri.

O objetivo de cada estudo de desempenho é tanto comparar diferentes alternativas ou encon-

trar o valor de parâmetro ótimo. Modelos analíticos geralmente fornecem o melhor discernimento dos efeitos de vários parâmetros e suas interações. Com simulações, isto pode ser possível para buscar o espaço dos valores de parâmetro para buscar uma combinação ótima, no entanto, frequentemente isto não é claro qual *trade-off* será entre diferentes parâmetros. Medição é a técnica menos desejável nesta situação. Não é fácil informar se o desempenho melhorado é o resultado de alguma mudança aleatória no ambiente ou é devido a alguma configuração de parâmetro em particular.

Segundo Kurian (2002), embora sejam classificadas em categorias diferentes, ferramentas e técnicas de modelagem e medição devem possuir algumas características desejáveis em comum, como:

- Devem ser não-intrusivos: o processo de medição não pode alterar o sistema ou degradar o desempenho do sistema;
- Não podem ser caros: a construção da unidade de medição de desempenho não deve possuir um custo significativo de tempo ou dinheiro;
- Devem ser fáceis de modificar ou extender: microprocessadores e sistemas computacionais constantemente sofrem alterações, e por este motivo deve ser fácil de extender a unidade de medição/modelagem para incluir o sistema atualizado;
- Não devem necessitar de código fonte das aplicações: se ferramentas e técnicas necessitam de código fonte, pode não ser possível avaliar aplicações comerciais onde o código fonte não está disponível;
- Devem ser rápidos: se um modelo de desempenho é muito lento, cargas de trabalho de longas durações que levam horas para executar, podem levar dias ou semanas para executar neste modelo. Além disso, se uma ferramenta de instrumentação é lenta, esta pode ser intrusiva;
- Devem ser amigáveis ao usuário: ferramentas que são difíceis de manusear frequentemente são sub-utilizadas. Ferramentas difíceis de usar também resultam em mais erros de usuário;
- Devem fornecer controle sobre os aspectos que estão sendo avaliados: isto deve ser possível para medir seletivamente o que é exigido.

Muitos desses requisitos são conflitantes. Por exemplo, é difícil para um mecanismo ser rápido e preciso ao mesmo tempo. Considerando modelos matemáticos, estes são rápidos, no entanto existem algumas hipóteses simplificadoras que são definidas na sua criação e frequentemente estas não são precisas. Da mesma forma, é difícil para uma ferramenta ser não invasiva e de fácil utilização. Muitos usuários se agradam com interfaces gráficas de usuário (IGU), no entanto, a maior parte das ferramentas de simulação e instrumentação com IGUs são lentas e invasivas.

Mesmo que alguns dos requisitos mais importantes tenham sido citados anteriormente, nem todos são possíveis e nem úteis para algumas aplicações, dessa forma é importante também saber quais requisitos a avaliação deve atender. E para definir requisitos que a avaliação do sistema deve seguir, é necessário ter conhecimento das métricas que serão abordadas na avaliação.

3.3 Métricas de Desempenho

Como mencionado anteriormente, outra questão importante na avaliação de desempenho é a escolha das métricas de desempenho. Para cada estudo de desempenho, um conjunto de critérios ou métricas de desempenho devem ser escolhidos. E uma das formas de preparar este conjunto é listar os serviços oferecidos pelo sistema.

Para cada requisito de serviço do sistema, há vários possíveis resultados. O sistema pode desempenhar o serviço corretamente, incorretamente ou rejeitar desempenhar o serviço. Por exemplo, um *gateway* em uma rede de computadores oferece o serviço de enviar pacotes para os destinos específicos em redes heterogêneas. Quando um pacote é disponibilizado, este pode ser enviado corretamente, pode ser enviado para um destino errado, ou pode ser cancelado e não ser enviado a nenhum destino. Similarmente a este processo, uma base de dados oferece um serviço de resposta a consultas. Quando uma consulta é requisitada, esta pode responder corretamente, responder incorretamente ou ser cancelada e não responder a ninguém. A seguir estas situações serão explicadas, com base em (JAIN, 2008).

Se o sistema desempenha um serviço corretamente, seu desempenho é medido pelo tempo gasto para executar este serviço, a taxa em que o serviço é desempenhado, e os recursos consumidos enquanto executa o serviço. Para medir estes processos, existem métricas que são relacionadas diretamente com *tempo-taxa-recurso* que são, capacidade de resposta, produtividade e utilização. Por exemplo, a capacidade de resposta de um *gateway* de uma rede é medida pelo seu tempo de

resposta - o intervalo de tempo entre a emissão e a chegada bem sucedida de um pacote de dados. A produtividade é medida pelo rendimento - o número de pacotes transmitidos por unidade de tempo. Já a utilização apresenta uma indicação da porcentagem de tempo dos recursos do gateway que estão ocupados para um dado nível de carga. O recurso com a mais alta utilização é denominada de gargalo.

Se o sistema desempenha o serviço incorretamente, significa que um erro tem ocorrido. Assim, é útil classificar erros e determinar as possibilidades de cada classe de erro. Por exemplo, considerando ainda o caso do *gateway*, pode-se querer encontrar a probabilidade de erros de 1-bit, 2-bits e assim por diante. Pode-se também querer encontrar a probabilidade de um pacote ser parcialmente entregue.

Se um sistema não desempenha o serviço, é dito que falhou ou está indisponível. Da mesma forma que no modelo de serviço incorreto, também é útil classificar os modos de falha e determinar as probabilidades de cada classe. Por exemplo, o mesmo *gateway* pode ter possibilidade de estar indisponível de $0,01\%$ do tempo devido a falhas de processador e $0,03\%$ devido a falhas de software.

Para as métricas associadas com os três resultados, de serviço bem sucedido, de erro e de indisponibilidade, são também denominadas de velocidade, confiança e disponibilidade. Para cada serviço oferecido pelo sistema algum destes possui uma quantidade de métricas de velocidade, uma quantidade de métricas de confiança, e uma quantidade de métricas de disponibilidade. Como a maioria dos sistemas oferecem mais do que um serviço, a quantidade de métricas cresce proporcionalmente a estes serviços.

Para um projetista a nível de sistema, tempo de execução e rendimento são duas métricas de desempenho muito importantes. Tempo de execução geralmente é a medida de desempenho mais importante. Esta é o produto do número de instruções, ciclos por instrução e o período de *clock*. Já o rendimento de uma aplicação é mais importante especialmente em sistemas de servidores. Em servidores que fornecem serviço à indústria bancária, indústria aérea ou outras áreas similares que possuem um certo tratamento de concorrência, o que importa é a quantidade de transações que podem ser completadas em unidade de tempo. Tais servidores, tipicamente denominados de sistemas de processamento de transações utilizam transações por minuto como uma métrica de desempenho (KURIAN, 2002).

O rendimento em alguns sistemas geralmente aumenta assim que sua carga também aumenta.

Depois de uma certa quantidade de carga, o rendimento para de aumentar e estagna, em alguns casos este pode começar a diminuir. O rendimento máximo alcançável sob condições de cargas de trabalho ideais é denominada de capacidade nominal do sistema. Para redes de computadores a capacidade nominal é denominada de largura de banda ou *bandwidth* e é comumente expressada em bits por segundo. Frequentemente, o tempo de resposta no rendimento máximo é muito alto para ser aceitável. Em tais casos é mais interessante conhecer o rendimento máximo alcançável sem exceder o limite do tempo de resposta pré-especificado. Isto pode ser denominado de **capacidade utilizável** do sistema (JAIN, 2008).

Em muitas aplicações, a curva de rendimento ou de tempo de resposta é considerada como um ponto de operação ótimo. Este é o ponto para além do qual o tempo de resposta aumenta rapidamente em função da carga, mas o ganho de rendimento é pequeno. Antes da curva, o tempo de resposta não aumenta significativamente, no entanto, o rendimento cresce assim que a carga aumenta. O rendimento na curva é denominado de curva de capacidade do sistema. É comum também medir a capacidade em termos de carga, por exemplo, o número de usuários ao invés do rendimento.

Com base em Jain (2008), a seguir serão listadas algumas métricas comumente utilizadas em vários modelos de sistemas.

- tempo de resposta: este é definido como o intervalo de tempo entre uma solicitação do usuário e a resposta do sistema;
- tempo de retorno: especialmente utilizada em sistemas de lote (*batch stream*), para medir a capacidade de resposta, esta métrica é o tempo entre a submissão de um lote e a realização da saída deste. O tempo para ler a entrada também é incluído;
- tempo de reação: é definido como o tempo entre a submissão de uma requisição e o começo de sua execução pelo sistema;
- rendimento: esta é definida como uma taxa (requisições por unidade de tempo) em que os pedidos podem ser atendidos pelo sistema;
- eficiência: a proporção entre a capacidade utilizável e a capacidade nominal é denominada de eficiência. Por exemplo, se o rendimento de uma Local Area Network (LAN) de 100-Mbps é somente 85Mbps, sua eficiência é de 85%;

- utilização: a utilização de um recurso é medido como uma fração do tempo em que o recurso está ocupado servindo requisições. Assim, essa é a proporção entre o tempo ocupado e o tempo decorrido total sobre um dado período;

- tempo ocioso: este é o período em que o recurso não está sendo utilizado. Gerentes de sistema frequentemente estão interessados em equilibrar a carga de modo que nenhum recurso seja mais utilizado do que outro. Alguns recursos, tais como processadores, estão sempre ocupados ou ociosos, tal que suas utilizações em termos de proporção de tempo ocupado para tempo total faz sentido;

- confiança: a confiança de um sistema é comumente medida pela probabilidade de erros ou pelo tempo médio entre os erros;

- disponibilidade: a disponibilidade de um sistema é definida como a fração de tempo em que o sistema está disponível para requisições de serviço de usuário;

- custo/desempenho: esta métrica é comumente utilizada para comparar dois ou mais sistemas. O custo inclui as despesas de licenciamento de hardware/software, instalação, e manutenção ao longo de uma determinada quantidade de anos. O desempenho é medido em termos de rendimento sobre uma dada restrição de tempo.

Estabelecer as métricas de desempenho adequadas para o sistema em desenvolvimento ou já pronto, representa um papel importante no processo de avaliação de desempenho. Com base nestas métricas estabelecidas é que serão alcançados os resultados que se pretende obter ou descobrir os que não estão nos planos do projetista.

Como o foco deste trabalho é voltado para sistemas embarcados, serão destacadas algumas métricas que possuem um contexto relacionado com estes dispositivos.

3.3.1 Métricas para Sistemas Embarcados

Como nodos sensores sem fio são dispositivos embarcados e, dependendo da sua aplicação, podem possuir restrição de tempo na sua execução, o planejamento das métricas que serão utilizadas como base para o processo de avaliação, consequentemente terá estas tecnologias como foco.

Um sistema embarcado é uma combinação de hardware e software e, possivelmente, de outras partes mecânicas para desempenhar uma função específica. Estes sistemas são utilizados para

várias aplicações. Estas aplicações podem ser proativas ou reativas dependendo dos requisitos como interface, escalabilidade, conectividade dentre outros. Uma outra subclasse de sistemas são os sistemas embarcados de tempo-real. Um sistema de tempo-real possui restrições de tempo e seu desempenho é especificado em termos da habilidade de fazer cálculos ou tomar decisões em tempo hábil. Estes cálculos importantes possuem prazos para a sua conclusão (KHAN et al., 2009).

Um prazo perdido é tão ruim quanto um resultado ruim e os danos causados por este atraso vai depender da aplicação. No contexto de RSSFs, um sensoriamento em um ambiente de monitoramento de área ambiental para prevenir ou detectar incêndios que não corresponda ao prazo, pode resultar em uma solução atrasada e que possivelmente não possa remediar o problema.

No entanto, a comunidade de sistemas embarcados não se beneficia com os novos avanços das metodologias de software, assim como é para o desenvolvimento de hardware. Mesmo aplicando alguns métodos de engenharia de software no desenvolvimento de software embarcado, é considerado que o processo de desenvolvimento de software embarcado é insatisfatório (OLIVEIRA et al., 2008).

Sendo assim, as principais métricas utilizadas no projeto de sistemas embarcados possuem foco nas características físicas do dispositivo, tais como, desempenho, memória, energia, potência, tamanho e largura, guiados pelas restrições do modelo (CORREA et al., 2010). Portanto, considerando as características relacionadas com o nodo sensor, as métricas mais adequadas são as que possuem foco no processamento, na energia e no tempo de processamento.

3.4 Conclusão

Avaliação de desempenho é um processo importante para compreender as características de um dispositivo ou sistema computacional. Com o auxílio desta é possível identificar os possíveis defeitos e qualidades que podem ocorrer na execução de um sistema.

Um processo de avaliação é definido em algumas etapas, são elas, selecionar a técnica de avaliação mais apropriada para aquele sistema, escolher as métricas que correspondem aos objetivos desejados, e por fim selecionar as cargas de trabalho que serão utilizadas no teste.

As cargas de trabalho, geralmente tratadas por intermédio de *benchmarks*, representam uma parte importante da etapa de avaliação, principalmente considerando simulações relacionadas ao processamento.

Capítulo 4

Cargas de Trabalho

Medições de desempenho de sistemas computacionais envolvem monitorar o sistema enquanto este está sendo sujeito a cargas de trabalho em particular. A fim de desempenhar medições significativas, as cargas de trabalho devem ser cuidadosamente selecionadas.

Neste capítulo serão apresentadas as definições e características acerca de cargas de trabalho e *benchmarks*. Na Seção 4.1 é apresentada uma definição desta tecnologia e uma categorização sobre os tipos de *benchmarks*. A Seção 4.2 explica como é o processo de modelagem de uma carga de trabalho. Na Seção 4.3 são apresentados os modelos de *benchmark* mais conhecidos e mais utilizados por empresas e pesquisadores para realizar um processo de avaliação de desempenho. Por fim, a Seção 4.4 apresenta as considerações finais.

4.1 Definição

O desempenho de um sistema é influenciado pelas características de seus componentes de hardware e software bem como da carga que tem de processar. A análise da carga de trabalho interpreta um papel importante em todos os estudos onde os índices de desempenho de um sistema têm de ser determinado. De fato, tais índices estão diretamente relacionados com as cargas de trabalho e não podem ser expressos por quantidades independentes destes.

Avalição de desempenho utilizando *benchmark* é o método primário para medir o desempenho de uma máquina física real. Um *benchmark* envolve executar um conjunto de programas bem definidos em uma máquina de teste para medir o desempenho. Vários *benchmarks* têm sido desenvolvidos para representar cargas de trabalho comuns (KRISHNASWAMY; SCHERSON, 2000).

Uma vez que o comportamento de uma carga de trabalho real é muito complexa e díficil de reproduzir, torna-se necessário utilizar um modelo para esta função. Tal modelo tem de capturar o comportamento dinâmico e estático da carga de trabalho real além disso, este deve ser compacto, repetitivo e preciso (CALZAROSSA; SERAZZI, 1993).

4.1.1 Categorização

O termo carga de trabalho de teste denota qualquer carga de trabalho utilizada em estudos de desempenho. Segundo Jain (2008) uma carga de trabalho pode ser real ou sintética. Uma carga de trabalho real é a carga observada em um sistema que está sendo utilizado por operações normais. Esta não pode ser repetida e portanto, não é adequada para uso como carga de trabalho de teste. Por outro lado, uma carga de trabalho sintética possui características que são similares as das cargas reais e podem ser aplicadas repetidamente de uma maneira controlada, além disso é desenvolvida e utilizada para estudos.

A principal razão para utilizar uma carga de trabalho sintética é que esta é uma representação ou modelo de uma carga de trabalho real. Além disso, outras razões são que os dados não são estritamente do mundo real, podendo ser grandes e conter dados sensíveis (JAIN, 2008).

Outra categoria de modelos de *benchmarks* comumente utilizada, são os *benchmarks* de kernel. Estes são baseados no conhecimento que 10% do código utiliza 80% dos recursos da CPU. Analistas de desempenho têm extraído estes fragmentos de código e tem utilizado-os como *benchmarks* (DIXIT, 1993).

No entanto, para Krishnaswamy e Scherson (2000) um *benchmark* pode ser categorizado conforme a granularidade da sua avaliação. Podendo ser classificado em duas categorias, *benchmarks* de baixa granularidade e *benchmarks* de alta granularidade.

A granularidade de um modelo de *benchmark* é determinada pela propriedade que pode ser medida utilizando os resultados do *benchmark* testado. Por exemplo, um *benchmark* medindo o desempenho das operações de uma única máquina é considerado como baixa granularidade, enquanto que um *benchmark* medindo o tempo de execução de grandes aplicações é considerado como alta granularidade (KRISHNASWAMY; SCHERSON, 2000).

Os *benchmarks* de aplicações e kernel são de alta granularidade devido aos resultados consistirem de tempos de execução de códigos grandes. A saída desses *benchmarks* é tipicamente um vetor de valores de MFLOP/S, onde cada elemento do vetor representa a taxa de execução de um

kernel ou aplicação no *benchmark* (KRISHNASWAMY; SCHERSON, 2000).

Alguns exemplos de *benchmarks* de alta granularidade populares incluem, *Linpack* para *benchmarks* de kernel, *Coremark* para *benchmarks* sintéticos, *Perfect Suite* e *SPEC* para *benchmarks* de aplicação.

Nos *benchmarks* de alta granularidade as aplicações/kernels pertecentes ao conjunto de *benchmark* aproxima-se da mistura representativa das cargas de trabalho do mundo real. Estes tipos de *benchmarks* possuem algumas deficiências. Os resultados dos *benchmarks* são úteis para medir o desempenho do sistema voltado para o usuário, mas fornece pouca informação para o projetista do sistema. Correlacionar medições de aplicação/kernel individuais em um *benchmarks* é complicado. Também, não é fácil explicar o desempenho medido baseado nas propriedades da arquitetura do sistema.

Já os *benchmarks* de baixa granularidade medem a taxa de execução de operações específicas diferentemente das aplicações ou kernels inteiros. Estas operações podem ser operações inteiras Operações Por Segundo(OPS), operações de ponto flutuante, operações de memória, operações de comunicação interprocessador, dentre outras.

Há uma quantidade de *benchmarks* que medem o tempo necessário para as operações específicas executarem kernels curtos para testarem cada operação. Os resultados do *benchmark* são utilizados para identificar gargalos no sistema, explicar a adequabilidade do sistema para uma certa aplicação, e fornecer informações sobre o sistema e a aplicação. No entanto, esta categoria de *benchmark* não é uma boa aproximação das cargas de trabalho reais do usuário, porque estes usam kernels criados artificialmente (KRISHNASWAMY; SCHERSON, 2000).

4.2 Modelagem de Carga de Trabalho

Embora a execução de uma carga de trabalho seja um fenômeno tipicamente determinístico, este é frequentemente modelado como um não-determinístico por intermédio das técnicas estatísticas da aplicação. Isto acontece por causa da grande quantidade de dados que são medidos e do grande número de variáveis interagindo entre si.

Para Calzarossa e Serazzi (1993) os principais passos para a construção de modelos de carga de trabalho, podem ser resumidos da seguinte forma:

1. Formulação:

Representa o nível de caracterização e o componente básico da carga de trabalho (ou seja, o menor nível de detalhes, tais como etapa de trabalho, comando interativo, transações de pesquisa/atualização de uma base de dados), junto com os parâmetros a serem usados por sua descrição, são selecionados. Um critério para avaliação do modelo de representatividade é também determinado.

2. Coleção dos parâmetros da carga de trabalho a serem modelados enquanto esta é executada.

3. Análises estatísticas dos dados medidos. Possui sub-etapas:

   (a) Análise preliminar:

   Os dados coletados podem incluir populações de cargas de trabalho distintas (por exemplo, compartilhamento de tempo, transação). A descoberta de partições naturais promove idéias por meio do foco em partes menores e mais gerenciáveis do conjunto de dados originais.

   (b) Análise das distribuições de parâmetro:

   Este tipo de análise pode conduzir à aplicação de vários tipos de transformações (por exemplo logarítmica), dos valores originais dos parâmetros ou para a identificação e a remoção de anexos.

   (c) Amostragem:

   Com o intuito de processar os dados medidos com uma quantidade moderada de tempo de processamento e de armazenamento, o número de componentes a ser considerado têm de ser pequeno. Portanto, uma amostra esboçada a partir dos dados medidos, é criada para ser utilizada nos passos subsequentes.

   (d) Análise estática:

   Uma classificação robusta é obtida quando os valores dos parâmetros são escalados de tal forma que estes se posicionam em um intervalo comum, ou seja, o particionamento das informações em grupos homogêneos de dados.

   (e) Análise dinâmica:

   Quando as características de variação de tempo da carga de trabalho (por exemplo, características dinâmicas das requisições de processamento, evolução no tempo das misturas dos componentes em execução), têm de ser reproduzidas, as propriedades de

várias séries temporais são consideradas. A aplicação de várias técnicas, tais como análise estatística de série de eventos não estacionários e processos estocásticos, fornecem uma representação concisa das sequências de dados analisados.
Esta análise pode ser definida como uma avaliação dinâmica correspondente a várias situações que podem ocorrer em um determinado sistema.

4. Representatividade:
   Vários critérios podem ser adotados para avaliar a representatividade de um modelo de carga de trabalho. O desempenho orientado a critérios, baseado na caracterização em termos de um vetor P, do qual os componentes são índices de desempenho selecionados de acordo com o objetivo de estudo, é amplamente aplicado. A precisão de um modelo de carga de trabalho é então medida como uma função da diferença entre P e P', obtido executando respectivamente a carga de trabalho real e seu modelo.

Mesmo considerando que estas formem uma sequência de passos comuns na definição de um modelo de carga de trabalho, estes modelos são designados para avaliar sistemas diferentes, com modelos de funcionamento diferentes, e muitas vezes, aplicados em sistemas computacionais distintos.

4.3 Modelos de *Benchmarks*

Atualmente, existem vários modelos de *benchmarks* que podem ser empregados em muitas tecnologias distintas. Há tanto modelos desenvolvidos por pesquisadores para auxiliar na avaliação de suas próprias pesquisas, como existem modelos disponibilizados por empresas ou organizações sem fins lucrativos para avaliar processadores ou outros dispositivos mais comuns aos usuários.

Nesta seção serão abordados os modelos de *benchmark*, mais citados e utilizados pelos pesquisadores e empresas. Os modelos voltados para avaliação de nodos sensores sem fio, ou mesmo da rede de sensores, serão apresentados no próximo capítulo.

Dentre os modelos de *benchmark* que serão apresentados a seguir, alguns são mais conhecidos por servirem como base para o desenvolvimento de *benchmarks* posteriores, e outros são conhecidos pelas organizações que os fomentam e por possuírem constantes atualizações.

4.3.1 SPEC

O Standard Performance Evaluation Corporation (SPEC) é uma corporação sem fins lucrativos formada para estabelecer, manter e endorsar um conjunto padronizado de *benchmarks* relevantes que podem ser aplicados às mais recentes gerações de computadores de alto desempenho. SPEC desenvolve conjuntos de *benchmarks* e também revisa e publica resultados submetidos pelos seus membros e outros licenciados (SPEC, 2014).

O desempenho de um sistema computacional nem sempre pode ser caracterizado por um único *benchmark*. Além disso, os fabricantes de dispositivos computacionais não chegaram a um acordo sobre um conjunto de *benchmarks* padrão, o que tornou praticamente impossível para os usuários avaliarem e compararem os sistemas disponíveis no mercado.

O grupo SPEC visou esses problemas selecionando e desenvolvendo *benchmarks* de aplicações reais que simulam os principais componentes do sistema. Seu objetivo é comparar desempenho de sistema através de diferentes tecnologias, arquiteturas, implementações, sistemas de memória, subsistemas de entrada e saída, sistemas operacionais, clocks, barramentos, compiladores, bibliotecas e aplicações de software (DIXIT, 1993).

Como o grupo SPEC desenvolve conjuntos de *benchmarks* para tecnologias distintas, existem diversos modelos desenvolvidos por este para utilizar na avaliação. Atualmente, os conjuntos de *benchmarks* que o SPEC (2014) fornece são:

- CPU:
  SPEC CPU2006 - projetado para fornecer medições de desempenho que podem ser utilizados para comparar cargas de trabalho intensivas em diferentes sistemas computacionais;

- Desempenho de estação de trabalho e gráficos:
  SPECviewperf 12 - este modelo mede desempenho de sistemas gráficos 3D que executam sob interfaces de programação em OpenGL e Direct X;
  SPECwpc - mais de 30 estações de trabalho são incluídos neste modelo para testar CPU, gráficos, E/S e largura de banda de memória;

- Computação de alto desempenho, OpenMP, MPI:
  SPEC HPC2002 - modelo usado para avaliar computação de alto desempenho de sistemas computacionais, foi retirado do programa em 2007, no entanto seus resultados ainda estão disponíveis para propósito histórico;

SPEC OMP2012 - projetado para medir desempenho usando aplicações baseadas no padrão Open Multi-Processing (OpenMP) 3.1 para processamento paralelo de memória compartilhada;
SPEC MPI2007 - modelo utilizado para avaliar Message Passing Interface (MPI), desempenho computacional, paralelo e de ponto flutuante em um amplo leque de clusters e hardware Symmetric MultiProcessing (SMP);

- Java client/server:
  SPECjbb2013 - este modelo tem sido desenvolvido para medir o desempenho baseado nas características de aplicações Java;

- Servidores de email:
  SPECmail2009 - todos os modelos para esta categoria foram retirados;

- Sistema de arquivo de rede:
  SPECsfs2008 - projetado para avaliar a velocidade e capacidade de controle de requisições de servidores de documentos;

- Medição de Energia:
  SPECpower ssj2008 - este modelo é o primeiro *benchmark* padrão de mercado que avalia a potência e características de desempenho de computadores servidores;

- Session Initiation Protocol (SIP):
  SPECsip infrastructure2011 - modelo projetado para avaliar a habilidade de um sistema de atuar como servidor SIP suportando uma aplicação SIP particular. A aplicação modelada é uma implementação para uma empresa de telecomunicação, ou provedor de serviços;

- Service Oriented Architecture (SOA):
  SOA - spec tem formado um novo subcomitê para desenvolver métodos padrões de medição de desempenho para middleware, base de dados e implementação de hardware de aplicações baseado em arquitetura orientada a serviço;

- Virtualização:
  SPECvirt sc2013 - SPEC atualizou este *benchmark* objetivando avaliação de desempenho de servidores de centro de dados usados na consolidação de servidores virtualizados;

- Servidores web:

  SPECweb2009 - todos os modelos para esta categoria foram retirados.

4.3.2 Coremark

O grupo Embedded Microprocessor *Benchmark* Consortium (EEMBC) desenvolve modelos de *benchmarks* para auxiliar desenvolvedores de sistemas a selecionar processadores mais otimizados para suas aplicações. Este desenvolve vários conjuntos de *benchmarks* visando diversas áreas computacionais como, automotiva, mídia digital, java, processadores multicore, redes, automação de escritórios, processamento de sinais, tablets/smartphones e browsers(EEMBC, 2014).

O coremark é um *benchmark* livre desenvolvido em 2009 pela EEMBC, com o intuito de substituir um modelo de *benchmark* mais antigo conhecido como Dhrystone para medir o desempenho de processadores. O coremark se vincula a um indicador de desempenho para execução de código simples, no entanto, ao invés de ser totalmente arbitrário e sintético, este código para *benchmark* usa estruturas de dados básicos e algoritmos que são comuns em praticamente qualquer aplicação. Além disso, o EEMBC escolheu cuidadosamente a implementação do coremark de tal forma que todos os cálculos são orientados por valores fornecidos em tempo de execução para evitar a eliminação de código durante a otimização de tempo de compilação. Coremark também estabelece regras específicas sobre como executar seu código e relatar os resultados, eliminando as inconsistências (GAL-ON; LEVY, 2010).

Sua estrutura é composta de listas, strings e matrizes. As listas comumente utilizam ponteiros e também são caracterizados por padrões de acesso de memória não serial. Em termos de testar o núcleo de uma CPU, o processamento da lista testa predominantemente quão rápido os dados podem ser usados para fazer a varredura atráves da lista. Este processamento consiste em inverter, buscar ou ordenar a lista de acordo com diferentes parâmetros baseados nos conteúdos dos dados da lista. Em particular, cada item da lista pode tanto conter um valor pré-calculado ou uma diretiva para invocar um algoritmo específico com dados específicos para fornecer um valor durante a ordenação.

Para verificar a operação correta, o coremark desempenha uma verificação de redundância cíclica de 16b baseada nos dados contidos em elemento da lista. Uma vez que o CRC é também uma função comumente utilizada em aplicações embarcadas, este cálculo é incluso na porção temporizada do coremark.

Geralmente o coremark é bem adequado para comparar processadores embarcados. É pequeno, altamente portável, fácil de compreensão e altamente controlado. Este verifica que todos os cálculos foram concluídos corretamente durante a execução, o que auxilia a depurar quaisquer problemas que possam surgir. As regras de execução são claramente definidas e as regras de reportação são aplicadas no próprio site do coremark. Além disso, o EEMBC oferece certificação para pontuação coremark e ainda possui um método padronizado para medir consumo de energia enquanto executa o *benchmark* (GAL-ON; LEVY, 2010).

4.3.3 Whetstone

O Whetstone consiste de um conjunto de onze módulos projetados para combinar frequências de operações utilizadas em programas ALGOL 949. Este *benchmark* exercita as características do processador como endereçamento de array, cálculo de ponto flutuante, chamadas de subrotinas, e passagem de parâmetros (JAIN, 2008).

Whetstone é uma das ferramentas de *benchmark* mais discutidas. Seus componentes básicos são um conjunto de programas ALGOL com ênfase em cálculos de ponto flutuante. Os parâmetros que caracterizam estes componentes são as freqüências relativas das instruções do compilador ALGOL Whetstone, medidos através de instrumentação. Isto resultou em uma população de tuplas componente-parâmetro-valor (CPV), que foram reduzidas pela média de combinações de instruções para produzir uma única tupla agregada, descrevendo as frequências das 25 instruções. Em seguida, a tupla CPV agregada foi substituída por um componente de carga de trabalho correspondente (PETERSON, 2010) (CURNOW; WICHMANN; SI, 1976).

Para construir a carga de trabalho de forma modular e controlada, foram utilizados oito módulos codificados a mão, representando os tipos básicos de computação. Para isso, foi utilizada uma técnica de busca para encontrar os pesos e as contagens de laços para os módulos, de modo que suas combinações de instruções correspondiam com a carga de trabalho agregada.

A intenção para o Whetstone ser independente de máquina é apoiar o uso de análise de alto nível, sendo que as vinte e cinco instruções modeladas representavam 95% do código de programa original. No entanto, os autores não conseguiram demonstrar que as instruções ignoradas não foram significantes do ponto de vista do desempenho, nem explicaram porque que as combinações de instruções, na época, foram necessárias e suficientes para representar com precisão o desempenho (PETERSON, 2010).

Em termos de métricas de precisão, o Whetstone é o mais próximo da carga original quanto possível (com um desvio médio de 15% com base nas combinações de instruções observadas), devido sua busca por erros nos módulos de coeficientes, serem especificamente minimizados (PETERSON, 2010).

Por fim, os resultados do Whetstone são medidos em Kilo Whetstone Instructions Per Second (KWIPS). Além disso, há muitas permutações existentes para este *benchmark*, tal que é importante garantir que comparações através de vários sistemas utilizem o mesmo código fonte e que o contador do laço interno seja definido grande o suficiente para reduzir a variação de tempo (JAIN, 2008).

4.3.4 Dhrystone

O modelo de *benchmark* Dhrystone foi criado em 1984 por Reinhold P. Weicker, até então membro da Siemens, com a intenção de medir o desempenho de sistemas computacionais. Devido à natureza dos sistemas da sua época, Weicker focou no desempenho de inteiros. Como o *benchmark* Whetstone para códigos de ponto flutuante já existia, o nome Dhrystone foi escolhido como sua contrapartida lógica, voltada para inteiros (WEISS, 2002).

A intenção original com o Dhrystone era criar um programa de *benchmark* pequeno que fosse representante de um sistema de programação de inteiros. O código Dhrystone é baseado em cálculos de inteiros simples, operações de string, decisões lógicas, e acessos de memória, projetados para refletir as atividades da CPU na maioria das aplicações de propósito geral. Os resultados do Dhrystone são determinados pela medição do tempo médio que um processador leva para desempenhar muitas iterações de um laço simples contendo uma sequência de instruções fixas que compõem o *benchmark*. Em relatórios, o Dhrystone é comumente referenciado em DMIPS, ou Dhrystone MIPS/MHZ (YORK, 2002).

Com relação as características do Dhrystone, algumas destas são positivas e outras negativas. Como esta é escrita em linguagem de programação C, permite que o código possa ser portado para um grande número de plataformas e arquiteturas. Seu tamanho pequeno, possibilita ao analista aprender rapidamente a controlar o Dhrystone. Em contra-partida, não se pode esperar que se iguale às aplicações com tamanho maior, com modos de medição mais complexos.

Outra característica sobre o Dhrystone é que este é um *benchmark* sintético que mede somente operações básicas e matemáticas. Além disso, somente codifica inteiros, o que torna-se potenci-

almente útil para microcontroladores simples de 8 e de 16 bits, no entanto, não mede pontos flutuantes, Single Instruction Multiple Data (SIMD), ou qualquer outro tipo de operação.

4.3.5 Linpack

O *benchmark Linpack* é uma coleção de subrotinas Fortran que foi originalmente projetado para auxiliar usuários do pacote *Linpack*, fornecendo informação no tempo de execução necessário para solucionar um sistema de equações lineares. O primeiro relatório do *Linpack* apareceu como um apêndice em um guia sobre este mesmo *benchmark* em 1979, que continha dados para um *path* comumente utilizado no pacote do *Linpack*. Os resultados eram fornecidos por um problema de matriz de tamanho 100, em um conjunto de 23 computadores testados. Isto foi realizado de forma que os computadores pudessem estimar o tempo necessário para solucionar seu problema de matriz por método de extrapolação (DONGARRA; LUSZCZEK; PETITET, 2003).

Ao longo dos anos, um desempenho adicional de dados foi inserido, assim como o escopo do *benchmark* também se expandiu. O relatório do *benchmark* descreve o desempenho pela solução de um problema de uma matriz densa $Ax = b$ em três níveis de problemas e otimizações, problema 100 por 100 (otimização de laço interno), problema 1000 por 1000 (otimização com três laços do programa inteiro), e um problema paralelo escalável (DONGARRA; LUSZCZEK; PETITET, 2003).

A idéia por trás do *Linpack* é, dado um problema envolvendo uma matriz A, então fatora-se e ou decompõe A em um produto simples de matrizes bem estruturadas que podem ser facilmente manipuladas para solucionar o problema original. O pacote tem a capacidade de controlar muitos tipos diferentes de matrizes e diferentes tipos de dados, e fornece um leque de opções.

Solucionar um sistema de equações necessita de $O(n^3)$ operações de ponto flutuante, mais especificamente, $2/3n^3 + 2n^2 + O(n)$ de somas e multiplicações de ponto flutuante. Assim, o tempo necessário para solucionar tais problemas em uma dada máquina pode ser aproximadamente:

$$time_n = \frac{time_{100} * n^3}{100^3} \tag{4.1}$$

O valor 100 na equação representa o tamanho da matriz, pois este foi o valor inicialmente utilizado devido às limitações de memória dos computadores da época. A matriz possuía $O(100^2)$ elementos de ponto-flutuante e era considerado um problema grande o suficiente para a época, sempre assumindo que eram pontos flutantes de 64 bits. Além disso, o algoritmo é baseado

em decomposição LU com pivotamento parcial. O tipo da matriz é real, geral e denso, com os elementos da matriz aleatoriamente distribuídos entre -1 e 1.

4.4 Conclusão

Para avaliar o desempenho de sistemas computacionais torna-se necessário avaliar a carga de trabalho ao qual aquele sistema está sendo ou será submetido. Esta é a função de alguns *benchmarks*, fornecer uma carga de trabalho para o sistema em questão.

No entanto, para alguns destes sistemas esta carga de trabalho deve ser simulada, seja de forma sintética, ou de forma fiel ao funcionamento do sistema.

Para ter conhecimento de qual abordagem utilizar, deve-se conhecer o processo de modelagem de cargas de trabalho e reconhecer qual é a mais adequada, considerando o funcionamento e a necessidade de aproximação da simulação com a realidade.

Por fim, a utilização de modelos já disponibilizados no mercado, que garantem uma avaliação mais apropriada de um sistema em questão, ou o desenvolvimento de novos modelos por pesquisadores para contrapor alguns modelos já ultrapassados ou inexistentes para algumas tecnologias, caracterizam uma etapa importante no processo de avaliação de desempenho.

Capítulo 5

Avaliação dos Nodos Sensores

Este capítulo descreve a implementação do *benchmark Linpack* em linguagem de programação NesC, e o processo de avaliação de desempenho dos nodos sensores. A Seção 5.1 apresenta uma contextualização do desenvolvimento do capítulo. A Seção 5.2 descreve alguns modelos de *benchmark* no contexto de redes de sensores sem fio. A Seção 5.3 apresenta os componentes utilizados na realização da pesquisa. A Seção 5.4 descreve o processo de implementação do *benchmark Linpack*, assim como o processo avaliação dos nodos e os resultados desta avaliação. A Seção 5.5 apresenta as considerações finais.

5.1 Introdução

Redes de sensores sem fio é uma tecnologia que vem sendo bastante estudada, com isso, novas plataformas de nodos sensores comerciais e acadêmicos surgem ano após ano, com modelos mais avançados e constantemente em desenvolvimento.

O processo de uma avaliação de desempenho de um sistema computacional é um trabalho minucioso. Cada avaliação necessita de um conhecimento detalhado do sistema que está sendo modelado, e deve ser feita uma seleção cuidadosa da metodologia, da carga de trabalho e das ferramentas a serem utilizadas (JAIN, 2008). O mesmo acontece no processo de avaliação de uma rede de sensores sem fio.

O *benchmark* que estará executando a carga de trabalho, deve gerar uma carga que deva ser adequada ao sistema em análise. Mesmo que seja um *benchmark* sintético, sua simulação deve ser a mais próxima possível da realidade que aquele sistema computacional estará executando.

Sendo assim, a medida que novos modelos de sensores vão surgindo, as metodologias já existentes para avaliar tais modelos vão se modernizando ou vão sendo substituídas por outras metodologias mais eficientes. Com essa diversidade de arquiteturas, torna-se difícil o desenvolvimento de um modelo de *benchmark* que aborde todas as características e funcionalidades para a realização de testes (HEMPSTEAD; WELSH; BROOKS, 2004). Contudo, como esta é uma tecnologia ampla, vários modelos de *benchmarks* surgem para avaliar não apenas o desempenho do nodo sensor, mas também para avaliar outras características da rede como um todo, levando em consideração as métricas adequadas para o processo de avaliação em questão.

Na seção a seguir, serão apresentados alguns dos modelos e técnicas de *benchmark* utilizados em redes de sensores sem fio.

5.2 Modelos de Benchmark para Redes de Sensores Sem Fio

Pesquisas utilizando *benchmark* em redes de sensores sem fio podem ter foco nas simulações de aplicações reais como em Nygren e Carlsson (2011), onde os autores fundiram o *Prowler*, um simulador probabilístico para redes de sensores sem fio, com o *SimuLink*, uma implementação do modelo de *benchmark* de simulação número 1 (*BSM1*), com o intuito de simular aspectos relevantes de uma rede de sensores e os efeitos de implementar estas no controle automático de tratamento de desperdício de água.

No entanto, as pesquisas mais comuns estão relacionadas com o desenvolvimento de novos modelos como, Gerwen et al. (2011), que desenvolveu um *benchmark* baseado em fluxo de trabalho, orientado para a avaliação de protocolos e aplicações em redes de sensores sem fio, onde o foco principal da pesquisa, é incentivar o desenvolvimento de modelos padronizados que auxiliem projetistas e usuários na escolha de protocolos e aplicações.

Em Hempstead, Welsh e Brooks (2004), este propõe o *TinyBench*, um conjunto de *benchmarks* padronizados para TinyOS, com foco em um *benchmark* para avaliação com estresse e um *benchmark* para aplicações reais. Considerando o fato que os dois *benchmarks* são focados no nodo sensor.

Além de modelos que avaliam as características da rede de sensores e do nodo sensor, existem os modelos que avaliam componentes de um nodo sensor, como microprocessadores. A pesquisa realizada por Nazhandali, Minuth e Austin (2005), apresenta um conjunto de aplicações para

representar cargas de trabalho de tempo-real, além de novas métricas de desempenho adequadas para avaliar microprocessadores de sensores sem fio.

A pesquisa realizada por Pesovic et al. (2012), é voltada para avaliação de microprocessadores de nodos sensores. Este avalia e compara os microprocessadores comumente utilizados em nodos sensores, como *ATMega128* da *Atmel*, *MSP430* da *Texas Instruments*, e os microprocessadores da série *Cortex-M* e *ARM7*, os dois da empresa *ARM*. No processo de avaliação foram utilizados os *benchmarks* sintéticos Dhrystone e Whetstone, e o *benchmark* de aplicação FIR (*Finite Impulse Response*).

Contudo, o foco da pesquisa é avaliar estes modelos de microprocessadores nos modos de operações ativo e *sleep*, e observar dentre os modelos, qual o mais eficiente de energia, considerando algumas métricas de desempenho como, consumo de energia e tempo de execução.

Estas pesquisas consolidam a motivação para este trabalho. Primeiramente, porque avaliação de desempenho de microprocessadores se dá por intermédio do uso de *benchmarks*. Segundo, porque quando se trata de avaliação de desempenho utilizando *benchmarks* em redes de sensores sem fio, o foco geralmente é voltado para a rede como um todo, embora em alguns casos, o foco resida no nodo sensor.

5.3 Componentes Utilizados

Para realização desta pesquisa foram utilizados um conjunto de componentes, onde cada um destes possui uma funcionalidade importante.

5.3.1 Plataforma MICAz

O modelo de nodo sensor utilizado nesta pesquisa para realizar na implementação do *benchmark* e realizar os testes, foi o MICAz da Memsic. Cada nodo é capaz de roteamento multi-salto e alto-configuração.

O MICAz opera dentro da frequência 2.4 GHZ de banda e é compatível com IEEE 802.15.4. É projetado especialmente para redes de sensores embarcados profundamente com capacidade de taxa de dados de 250 kbps (ALI; DRIEBERG; SEBASTIAN, 2011).

A Figura 5.1 exibe um modelo de nodo sensor da plataforma MICAz, semelhante ao utilizado na pesquisa. Este mote possui em sua estrutura alguns subcomponentes como, memória flash

serial externa de 512kb, um identificador integrado (*Unique ID*), *CC2420*, que é o rádio específico para esta plataforma, um conector de 51 pinos e um microcontrolador de sinais analógicos e digitais, conhecido como ATmega128L.

Figura 5.1: Nodo sensor com a plataforma MICAz.

A Figura 5.2 demonstra a arquitetura do nodo sensor, a organização dos subcomponentes e como estes estão interligados.

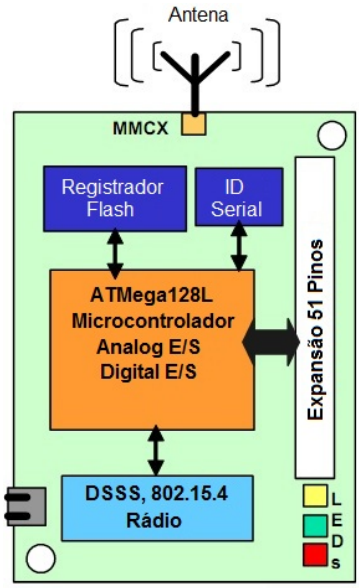

Figura 5.2: Arquitetura de hardware da plataforma MICAz. Fonte: (MEMSIC, 2010).

5.3.2 Microcontrolador ATmega128L

O ATmega128L da Atmel é um microcontrolador, utilizado em aplicações embarcadas e equipado com interfaces externas relevantes para periféricos comuns (KARL; WILLIG, 2005). Além de executar aplicações de usuários e pilhas de protocolos *XMesh* (MEMSIC, 2010).

O ATmega128L é um microcontrolador Complementary Metal-Oxide-Semicondutor (CMOS) de 8 bits, baseado em microcontroladores AVR melhorados com arquitetura RISC. Por executar instruções em um único ciclo de clock, o ATmega128L alcança rendimento aproximando de 1 MIPS por MHz, permitindo ao desenvolvedor do sistema otimizar consumo de energia com base na velocidade de processamento.

O núcleo AVR combina um rico conjunto de instruções com 32 registradores trabalhando com propósito geral. Além disso, o ATmega128L fornece as seguintes características, 128 Kbytes de memória flash programável com capacidade de leitura enquanto escreve, 4 Kbytes de memória Electricallly-Erasable Programmable Read-Only Memory (EEPROM), 4 Kbytes de memória Static Random Access Memory (SRAM) e contador de tempo real.

Um processador possui tipicamente vários modos de operação que consomem diferentes quantidades de energia. No modo ativo completo, o processador consome a quantidade de energia máxima, que é tipicamente 8 mA. No modo de consumo de energia mais baixo, como o modo *sleep*, o processador consome aproximadamente 15 μAmps (MEMSIC, 2010).

5.3.3 TinyOS 2.1.2 e NesC

TinyOS é um sistema operacional *open source* (código livre) orientado a eventos e desenvolvido em linguagem de programação NesC. É projetado para dispositivos de comunicação sem fio com baixo consumo de energia, tais como os utilizados em redes sensores sem fio, que possuem recursos muito limitados (GAY D., 2009).

O TinyOS fornece abstrações de software dos hardwares básicos dos dispositivos. Por exemplo, o TinyOS pode apresentar chip de armazenamento de memória flash, que possui blocos e setores com propriedades de escrita e exclusão, como uma simples abstração de um registro circular. Fornecer abstrações de software úteis bem projetados e testados, simplifica muito o trabalho dos desenvolvedores de sistemas e aplicações (TINYOS, 2013).

No TinyOS, a aplicação de usuário não está ativamente ciente do gerenciamento de energia. Não há muitas chamadas ao gerenciamento de energia explícitas fornecidas pelo sistema operacional em tarefas como por exemplo, unidade de execução. Dessa forma, o escalonador irá por o processador para dormir, uma vez que a fila de tarefas esteja vazia (MEMSIC, 2010).

TinyOS utiliza a linguagem NesC ao invés da linguagem C, pois fornece suporte para modelo de programação e execução. Por exemplo, ao invés de utilizar macros para programar *constructs*

como tarefas e eventos, o uso do NesC permite que o compilador verifique a utilização destes. Além disso, o compilador NesC produz um código C para um compilador C subjacente que está formatado para maximizar o grau de otimização (TINYOS, 2013).

NesC é uma linguagem de programação para sistemas embarcados em rede, tais como os motes. NesC suporta um modelo de programação que integra concorrência, comunicação e reatividade para o ambiente. Por desempenhar otimizações no programa inteiro e detecção de corrida de dados em tempo de compilação, NesC simplifica o desenvolvimento de aplicações, reduz o tamanho de código, e elimina muitas fontes de erros (GAY et al., 2003).

Um dos principais focos de NesC é o design em sistema holístico. As aplicações do mote são profundamente ligadas ao hardware, e cada mote executa uma aplicação por vez. Esta abordagem produz três propriedades importantes. Primeiro, todos os recursos são conhecidos estaticamente. Segundo, em vez de empregar um sistema operacional de propósito geral, as aplicações são construídas a partir de um conjunto de componentes de sistema reutilizáveis acoplados com código específico de aplicação. Terceiro, o limite de hardware/software varia dependendo da aplicação e da plataforma de hardware, o que é importante para o projeto para a decomposição flexível (GAY et al., 2003).

A versão do TinyOS utilizado neste trabalho foi o TinyOS 2.1.2, pois esta era a versão mais atual no momento em que a fase de implementações nos motes foi iniciado. Como o sistema operacional escolhido para os motes foi o TinyOS, a linguagem selecionada para implementar o *benchmark* foi o NesC.

5.3.4 Arduino UNO

Arduino é uma plataforma computacional física *open source* baseada em uma placa simples de E/S (Entrada/Saída) e um ambiente de desenvolvimento que implementa uma linguagem de processamento. O arduino pode ainda ser utilizado para desenvolver objetos autônomos alternativos ou pode ser conectado ao software em um computador (BANZI; CANDUCCI; WALLACE, 2009).

Para Banzi, Canducci e Wallace (2009) o arduino se difere de outras plataformas de desenvolvimento devido a algumas características:

- É um ambiente multiplataforma, pode ser executado no Windows, IOS e Unix;
- É baseado em uma IDE de programação de processamento, um ambiente de desenvolvi-

mento fácil utilizado por projetistas;

- Pode ser programado via cabo USB e não serial. Esta característica é útil, devido a muitos computadores modernos não possuírem portas seriais.
- Seu hardware e software são *open source*. Se um desenvolvedor desejar, pode fazer o *download* do diagrama do circuito, comprar todos os componentes, e fazer a prória placa sem pagar aos desenvolvedores originais;
- Há uma comunidade ativa de usuários.

Para a etapa da pesquisa focada na execução de testes, foi utilizado um modelo de arduíno conhecido como arduino UNO. O arduino UNO é uma placa microcontroladora baseada no ATmega328. Este possui 14 pinos de entrada e saída, 6 entradas analógicas, um ressoador cerâmico de 16 MHz, conexão USB e um botão reset. O UNO difere-se das placas anteriores, pois este não utiliza chip serial-USB FTDI. Ao invés disso, este apresenta o ATmega16U2 previamente programado como um conversor serial-USB (ARDUINO, 2014).

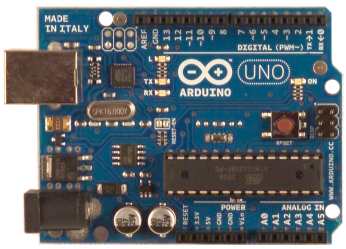

Figura 5.3: Exemplo de arduino UNO.

O arduino UNO da Figura 5.3, foi o modelo utilizado nesta pesquisa como um meio de fornecer energia para os nodos sensores e assim energizá-los com a quantidade de energia necessária para realizar o processamento em cada nível diferente de voltagem.

5.4 Avaliação de desempenho utilizando o Linpack

Para avaliar o desempenho dos nodos sensores, é necessário utilizar um *benchmark* para gerar cargas de trabalho e assim obter resultados para realizar a avaliação. Para gerar a carga de trabalho no nodo sensor e medir o desempenho foi utilizado um *benchmark* de kernel, pois como não é

uma avaliação para uma aplicação específica, o conhecimento da carga de trabalho real não se faz necessário.

5.4.1 Linpack

Nesta pesquisa foi implementado o *benchmark Linpack* para linguagem de programação NesC. O *benchmark Linpack* foi escolhido dentre um grupo de outros *benchmarks*, devido as suas características, sua arquitetura bem documentada e principalmente pela sua carga de trabalho.

A carga de trabalho do *Linpack* é dividida em duas partes. A primeira parte realiza a decomposição com pivotamento parcial de uma matriz com números reais gerados aleatoriamente. A segunda parte utiliza os resultados desse pivotamento para solucionar um conjunto de equações lineares.

Uma grande parcela do tempo de execução é gasto na segunda parte do algoritmo. Isso se deve ao fato que esta parte é dividida em mais três sessões. Uma delas é responsável por aproximadamente 90% do tempo de processamento gasto. Esta sessão é responsável por multiplicar um escalar α, vezes um vetor x e somar os resultados a outro vetor y. As outras duas sessões correspondem a solução das equações lineares (DONGARRA; LUSZCZEK; PETITET, 2003).

Com relação às características relacionadas ao *Linpack*, podem ser relacionadas:

- Possui modo de operação com versão de precisão simples para pontos flutuantes de 32-bits e precisão dupla para pontos-flutuantes de 64bits;
- Desenrolamento de laços - uma grande parcela do tempo gasto em processamento se deve aos laços internos no código fonte, assim desenrolando os laços, seu conteúdo é replicado ao menos uma vez;
- Estrutura do algoritmo dividida em duas partes, facilita a compreensão da base do algoritmo;
- Conjunto de subfunções para incrementar no processamento;
- Tamanho da matriz variável conforme a arquitetura do hardware do dispositivo em avaliação.

Embora estas características do *Linpack* sejam muito importantes para o seu funcionamento, nesta pesquisa implementamos apenas as características que possuem influência direta na geração da carga de trabalho do *benchmark*.

5.4.2 Processo de Avaliação

Para o desenvolvimento do processo de avaliação, primeiramente foram analisadas quais as métricas de desempenho que poderiam ser utilizadas. Como a leitura do processamento é um dos pontos principais da pesquisa, o cálculo de operações do processador foi a primeira métrica abordada. Contudo, o *benchmark Linpack* possui sua contagem de operações dadas em Millions Floating-point Operations Per Second (MFLOP/S). A partir desta situação, foi necessário um tratamento das informações no momento da implementação, pois essa contagem de operações no nodo sensor não seria adequado para a fase de análise dos resultados. Portanto, foi necessária na fase de implementação do algoritmo, a conversão da métrica de operações de MFLOP/S para Floating-point Operations Per Second (FLOP/S).

A segunda métrica escolhida para o processo, foi o tempo de execução. Esta métrica se fez necessária para avaliar o tempo que o processador leva para executar a carga de trabalho e avaliar se a alteração de energia influi neste tempo.

Após a seleção das métricas adequadas para a avaliação, foi definido como seria realizado este processo de avaliação. Primeiramente, foi selecionado arbitrariamente um nodo sensor do kit de motes da Memsic, isso porque estes possuem as mesmas características físicas. Posteriormente, foi inserido o código-fonte do *benchmark* por intermédio da placa de programação USB do kit, no nodo sensor.

Como o controle da quantidade de energia que o nodo sensor pode possuir é complicado de realizar manualmente, foi utilizado nesta fase o arduino UNO. Com o auxílio deste dispositivo e um circuito externo, foi possível controlar diretamente a energia para o nodo sensor. Para isso, foi desenvolvido um software em Java para seleção dos níveis de tensão e foi utilizado um software para conexão com a placa do arduino, tornando assim possível disponibilizar o nível de tensão de acordo com os níveis selecionados na aplicação.

Para esta etapa do processo de avaliação foi escolhido controlar os níveis de energia do nodo de forma direta, ao invés da utilização de pilhas. Isso se deve ao fato que o controle direto da energia disponibilizada ao nodo sensor, auxilia em uma observação mais ampliada e clara dos níveis de energia. A utilização de pilhas acarretaria em um processo de avaliação mais demorado e, possivelmente, mais complicado devida à necessidade de desgastar as pilhas em níveis de energia distintos.

A Figura 5.4 exibe o arduino conectado a uma fonte de energia externa por interface usb. O

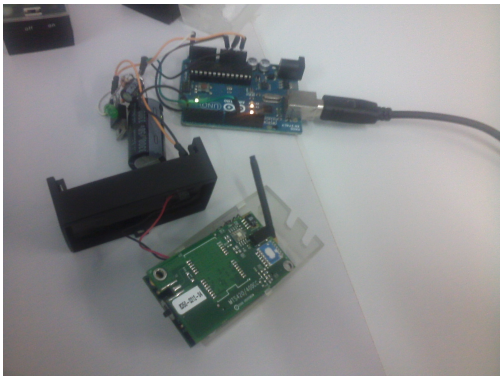

Figura 5.4: Imagem do nodo sensor sendo energizado pelo arduino.

arduino e o circuito externo auxiliam no fornecimento e controle dos níveis de tensão disponibilizados para o nodo sensor.

O processo de avaliação ficou definido da seguinte forma. O notebook utilizado no processo, é conectado ao arduino por interface usb, servindo também como fonte de energia. O arduino e o circuito externo são integrados ao nodo, auxiliando no fornecimento e controle de energia. O nodo sensor, por sua vez, executa o Linpack novamente ao término da execução anterior. A cada período de execução, são enviados os valores de processamento e de tempo de execução para a estação base, por intermédio de comunicação sem fio. A estação base envia estes dados para o notebook via interface usb. Como mencionado anteriormente, os níveis de tensão são determinados por uma aplicação Java executando no notebook, assim como outra aplicação Java para visualização dos dados advindos da estação base.

Para avaliação dos níveis de tensão, foram selecionados três abordagens de operação, oscilação dos níveis de tensão com valores entre $2,23$ e $2,64$, oscilação dos níveis de tensão com valores entre $2,67$ e $3,27$. E a terceira abordagem com níveis entre $2,29$ e $3,27$.

Como existem muitos valores de processamento e de tempo de execução relacionados com cada nível de tensão e, por sua vez, relacionados com as oscilações, foi realizado nesta etapa um processo de média de valores. Esta média corresponde a média de valores de tensão aproximados e os valores contidos nestas leituras. Junto a este processo de médias de valores e de tensões, foi realizado o processo de normalização das médias.

O processo de normalização se fez necessário devido aos valores de processamento e de tempo serem divergentes. Para isso foi utilizada uma fórmula de normalização para nivelar os valores das duas métricas.

$$N_i = \frac{x_i}{\bar{x}} \tag{5.1}$$

Onde N_i representa as instâncias normalizadas, x_i são os valores tabelados, $\bar{x}$ são as médias dos valores. Sendo $i = 1, 2, 3, ..., n$ as instâncias de voltagens.

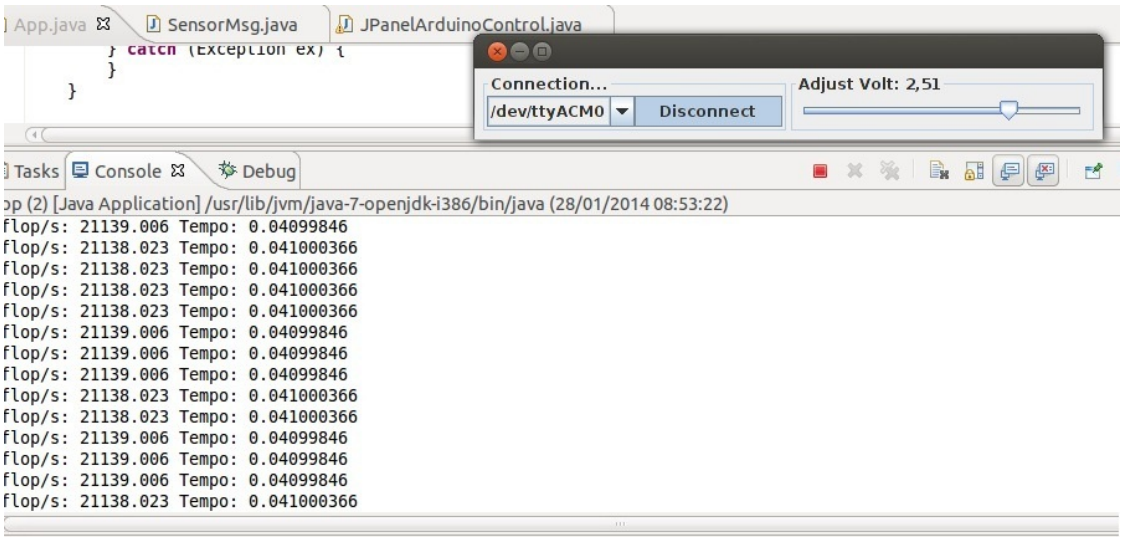

Figura 5.5: Imagem da aplicação com a interface de controle de energia.

A Figura 5.5 apresenta uma imagem com a interface de gerenciamento de energia, onde é possível visualizar o controle do ajuste de voltagem e a quantidade de energia disponibilizada ao nodo, além de várias leituras de processamento e de tempo de execução.

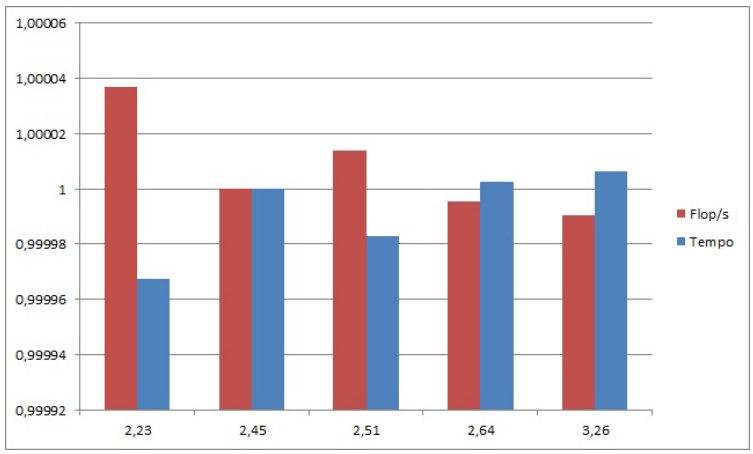

Figura 5.6: Avaliação do nodo sensor.

A Figura 5.6 apresenta um gráfico correspondente à normalização dos resultados de processamento e tempo de execução do nodo sensor na primeira abordagem. Com base nas informações

apresentadas neste gráfico é possível visualizar que, embora os níveis de tensão estejam se elevando, os valores de processamento e de tempo de execução se elevam e declinam. A média dos valores de processamento são mais elevados nos níveis de tensão mais baixos. Já nos níveis de tensão mais altos, a média dos valores de processamento foram mais baixos. Estes fatos não contestam a quantidade de energia que um nodo deve possuir, e sim que a quantidade de energia quando modificada influencia nas leituras de processamento.

Os valores utilizados no processo de média e normalização dos dados para geração do primeiro gráfico do nodo sensor, aparecem na Tabela 5.1.

Tabela 5.1: Tabela com os valores do processamento do nodo sensor.

Volts	Flop/s	Tempo
2,23	21139,17	0,040998
2,45	21138,39	0,040999
2,51	21138,68	0,040999
2,64	21138,29	0,040999
3,26	21138,19	0,041000

Com base nos dados da Tabela 5.1, observa-se que na contagem de FLOP/S a oscilação é perceptível nos décimos de FLOP/S, já no tempo a variação se dá em milésimos de milésimos de segundos.

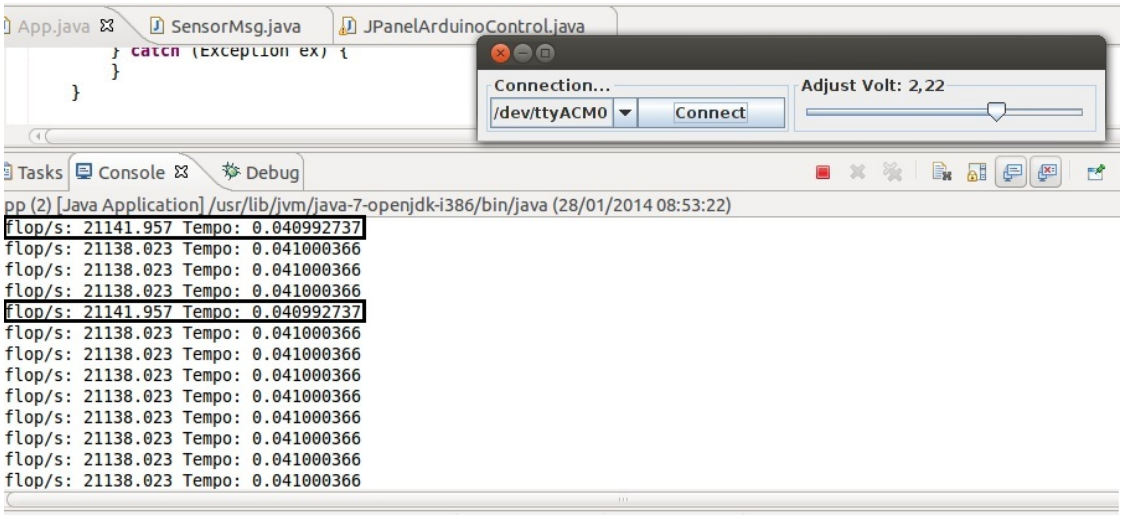

Figura 5.7: Imagem que exibe os resultados do processamento no nodo sensor.

As leituras de processamento da Figura 5.7, apresentam valores uniformes para a contagem de FLOP/S, o equivalente a 21138.023 FLOP/S. No entanto, em dois momentos de oscilação de

processamento essas leituras chegaram ao valor de 21141.957 FLOP/S. Considerando que o *benchmark* utilizado não reproduz uma carga de trabalho real, em uma aplicação onde a precisão dos resultados é muito importante, essa variação mesmo sendo pequena, pode influenciar na atividade desta aplicação.

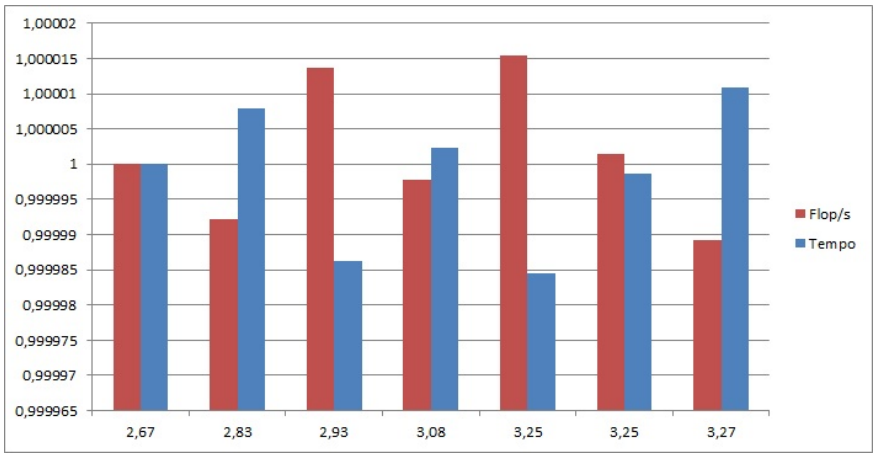

Figura 5.8: Avaliação do nodo sensor.

No gráfico da Figura 5.8, referente à segunda abordagem de operação, acontecem situações de oscilações semelhantes às que acontecem no gráfico da Figura 5.6. Inicialmente os valores de processamento e de tempo de execução são semelhantes, posteriormente os valores de processamento se elevam e declinam antes e depois dos 3,00 volts de voltagem. Embora as tensões 3,25, que se repete, e 3,27 sejam próximas, seus valores de processamento são diferentes.

Com base nas informações deste gráfico, é possível visualizar que assim como os primeiros testes realizados no nodo sensor, este gráfico não possui um padrão entre os níveis de processamento e de tempo relacionados com a voltagem.

Tabela 5.2: Tabela com os valores do processamento do nodo sensor.

Voltagem	Flop/s	Tempo
2,67	21138,55	0,040999348
2,83	21138,38	0,040999672
2,93	21138,84	0,040998787
3,08	21138,50	0,040999441
3,25	21138,87	0,040998716
3,25	21138,58	0,040999290
3,27	21138,32	0,040999794

Observando os valores da Tabela 5.2 percebe-se que, assim como nos primeiros testes as

variações de processamento se dão em décimos de FLOP/S, enquanto que as variações de tempo se dão em milésimos de milésimos de segundos.

A Figura 5.9 exibe uma sequência de leituras de processamento semelhantes aos primeiros testes realizados no nodo sensor, com valores equivalentes a 21138.023 FLOP/S. No entanto, pode-se perceber que dentre estas leituras de processamento, há duas leituras com valores diferentes, equivalentes a 21145.893 FLOP/S. A variação no processamento é de 7 FLOP/S, um pouco superior ao do primeiro teste realizado no nodo sensor.

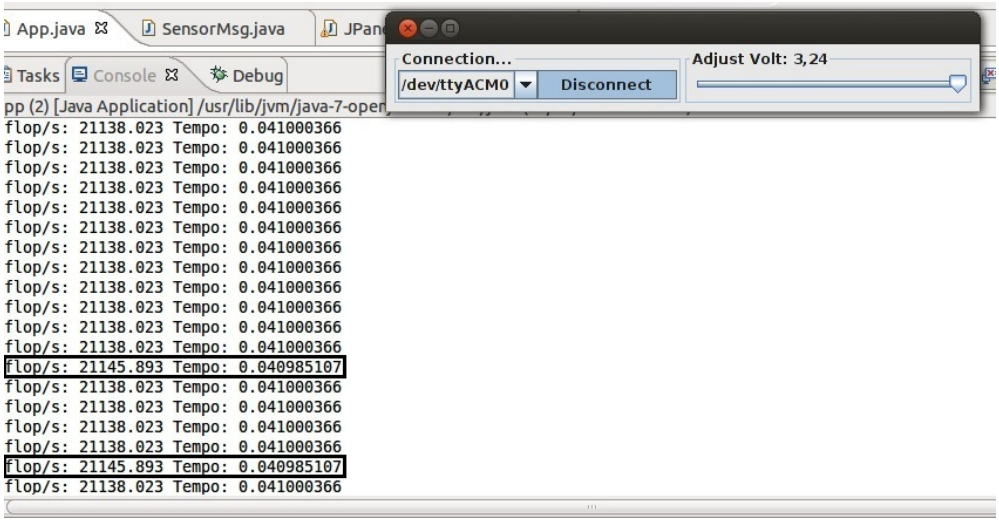

Figura 5.9: Imagem que exibe os resultados do processamento no nodo sensor.

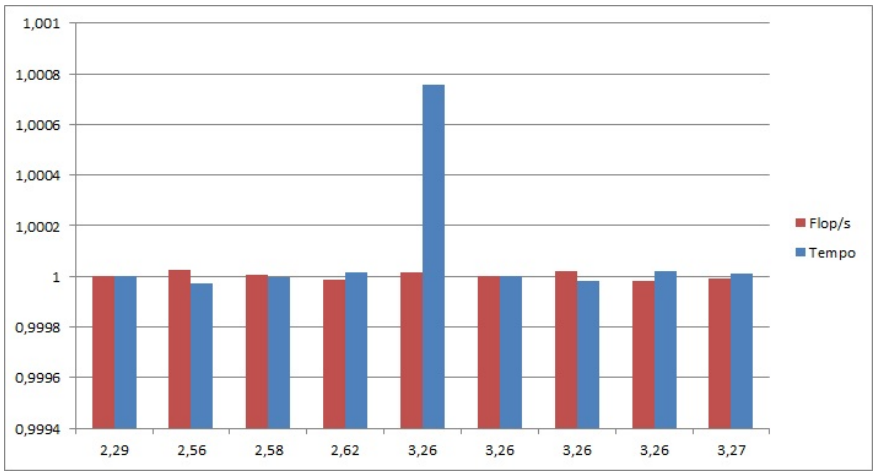

Figura 5.10: Avaliação do nodo sensor.

A Figura 5.10 apresenta o gráfico referente às leituras de processamento do nodo sensor na terceira abordagem. Este gráfico, diferente dos gráficos anteriores, possui níveis de variações

muito pequenos, mas isto se deve ao fato que em uma das instâncias a quantidade de valores na leitura de processamento foi mais elevado que nas outras instâncias, influenciando na geração do gráfico. Devido a isso, um dos valores é bem diferente dos outros apresentados no gráfico.

Assim como nas outras abordagens, não há uma padronização nas oscilações do teste realizado. Os valores de processamento e de tempo oscilam independente dos níveis de tensão serem altos ou baixos. Nesta abordagem, mais da metade dos níveis de tensão possuem valores acima de 3,00 volts. Embora possua quatro instâncias semelhantes, seus valores ainda oscilam, situação semelhante encontrada na segunda abordagem.

Tabela 5.3: Tabela com os valores do processamento do nodo sensor.

Voltagem	Flop/s	Tempo
2,29	21138,68	0,040999094
2,56	21139,22	0,040998043
2,58	21138,78	0,040998898
2,62	21138,39	0,040999645
3,26	21138,28	0,040999868
3,26	21139,09	0,040998304
3,26	21138,68	0,040999094
3,26	21138,99	0,041030153
3,27	21138,45	0,040999536

A Tabela 5.3 exibe os valores correspondentes aos valores de processamento do nodo sensor na terceira abordagem. Embora esta tabela apresente dois valores com variação de 1 FLOP/S, ainda assim a oscilação do processamento será perceptível em décimos de FLOP/S, enquanto que os níveis de tempo oscilam em milésimos de milésimos de segundos.

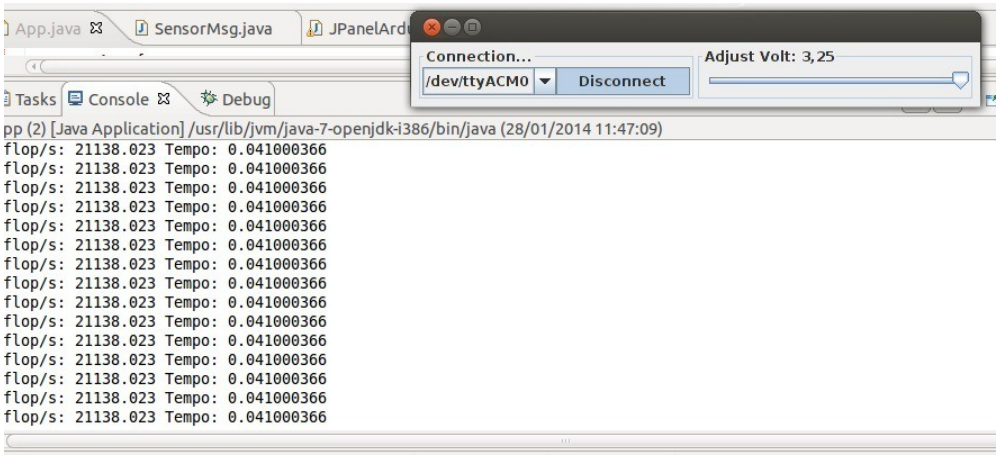

Figura 5.11: Imagem que exibe os resultados do processamento no nodo sensor.

Representando bem o equilíbrio de valores do gráfico da Figura 5.10, a Figura 5.11 apresenta um conjunto de leituras de processamento semelhantes.

5.5 Conclusão

Neste capítulo foram apresentadas as informações que contribuíram para a escolha do *benchmark Linpack* ser utilizado como ferramenta do processo de avaliação. Também foi apresentada a metodologia utilizada para realizar o processo de avaliação do nodo, incluindo o controle de energia manual com o auxílio do arduino. Por fim, foram exibidos imagens e gráficos com os resultados do processamento e do tempo de execução do nodo sensor, considerando três abordagens de níveis de energia.

Capítulo 6

Considerações Finais

Esta pesquisa teve como objetivo principal estudar e avaliar nodos sensores com o intuito de descobrir se a oscilação dos níveis de tensão influenciam nos valores de processamento e de tempo de execução do microcontrolador.

Para este processo, foi utilizado um esboço de fornecimento de energia para o nodo, de forma a auxiliar no controle direto dos níveis de tensão que são disponibilizados ao nodo. Este esboço é constituído por um notebook (fonte de energia), um modelo arduino UNO e um circuito externo auxiliar.

Como parte do processo de avaliação, foi implementado um modelo de *benchmark* conhecido como *Linpack*, com o intuito de auxiliar no processo de avaliação de desempenho do microcontrolador ATmega128L. O algoritmo foi construído tendo como base as características básicas da carga de trabalho do *benchmark Linpack*. Dessa forma, a contribuição se deu por intermédio da sua carga de trabalho e das suas métricas de desempenho.

Os gráficos do processo de avaliação apresentados nesta dissertação, estão normalizados e ordenados por níveis crescentes de tensão. As variações apresentadas nos gráficos, são significativas do ponto de vista da aplicação. Nodos sensores podem ser implementados em diversos ambientes, em especial, ambientes de difícil acesso. Estes ambientes podem influenciar no consumo de energia do nodo, influenciando assim no processsamento do microcontrolador.

Estima-se que esta pesquisa incite outros pesquisadores a avaliar os outros componentes do nodo sensor, abordando as métricas relacionadas com estes, de acordo com a energia consumida pelo nodo.

6.1 Contribuições

As contribuições desta dissertação são:

- Comprovação que as variações de energia em um nodo sensor influenciam no valor do processamento do microcontrolador;
- Implementação de um modelo de *benchmark* para nodos sensores sem fio.

6.2 Trabalhos Futuros

Com base neste trabalho alguns trabalhos futuros podem ser listados:

- Avaliar os outros componentes da arquitetura do nodo sensor, observando a influência que a redução ou acréscimo de energia pode acarretar em cada componente;
- Implementar as outras características do *benchmark Linpack*, considerando a adequação com sistemas embarcados;
- Avaliar os nodos, observando minuciosamente as causas de alteração de energia, com o intuito de encontrar um padrão para estas alterações de valores.

Referências bibliográficas

ALI, N.; DRIEBERG, M.; SEBASTIAN, P. Deployment of micaz mote for wireless sensor network applications. In: *Computer Applications and Industrial Electronics (ICCAIE), 2011 IEEE International Conference on*. [S.l.: s.n.], 2011. p. 303–308.

ANASTASI, G. et al. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Netw.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 7, n. 3, p. 537–568, maio 2009. ISSN 1570-8705. Disponível em: <http://dx.doi.org/10.1016/j.adhoc.2008.06.003>.

ARDUINO. *Arduino Board UNO*. 2014. Acessado em 27 de Janeiro de 2014. Disponível em: <http://arduino.cc/en/Main/arduinoBoardUno>.

BANZI, M.; CANDUCCI, E.; WALLACE, S. *Primeiros Passos com o Arduino*. Novatec, 2009. ISBN 9788575222904. Disponível em: <http://books.google.com.br/books?id=Ts-gpwAACAAJ>.

Borges Neto, J. B. *PROST: Um protocolo de roteamento sensível ao tempo para redes de sensores sem fio*. 141 p. Dissertação (Mestrado) — Universidade Federal do Ceará, Fortaleza, 2009.

BOUDEC, J.-Y. L. *Performance Evaluation of Computer and Communication Systems*. [S.l.]: EPFL Press, Lausanne, Switzerland, 2010. 420 p. ISBN 978-2-940222-40-7.

CALZAROSSA, M.; SERAZZI, G. Workload characterization: a survey. *Proceedings of the IEEE*, v. 81, n. 8, p. 1136–1150, 1993. ISSN 0018-9219.

CASALE, G.; GRIBAUDO, M.; SERAZZI, G. Tools for performance evaluation of computer systems: historical evolution and perspectives. In: *Proceedings of the 2010 IFIP WG 6.3/7.3 international conference on Performance Evaluation of Computer and Communication Systems: milestones and future challenges*. Berlin, Heidelberg: Springer-Verlag, 2011. (PERFORM'10), p. 24–37. ISBN 978-3-642-25574-8. Disponível em: <http://dx.doi.org/10.1007/978-3-642-25575-5-3>.

CHARFI, W. et al. A behavioural study of nodes to conserve energy in wireless sensor networks. In: *Design and Technology of Integrated Systems in Nanoscale Era (DTIS), 2010 5th International Conference on*. [S.l.: s.n.], 2010. p. 1–6.

CORREA, U. et al. Towards estimating physical properties of embedded systems using software quality metrics. In: *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*. [S.l.: s.n.], 2010. p. 2381–2386.

CURNOW, H. J.; WICHMANN, B. A.; SI, T. A synthetic benchmark. *The Computer Journal*, v. 19, p. 43–49, 1976.

DARGIE, W.; POELLABAUER, C. *Fundamentals of Wireless Sensor Networks - Theory and Practice*. [S.l.]: John Wiley and Sons, 2010. ISBN 978-0-470-99765-9.

DIXIT, K. M. Overview of the spec benchmarks. In: GRAY, J. (Ed.). *The Benchmark Handbook*. Morgan Kaufmann, 1993. ISBN 1-55860-292-5. Disponível em: <http://people.cs.uchicago.edu/ chliu/doc/benchmark/chapter9.pdf>.

DONGARRA, J. J.; LUSZCZEK, P.; PETITET, A. The linpack benchmark: Past, present, and future. concurrency and computation: Practice and experience. *Concurrency and Computation: Practice and Experience*, v. 15, 2003.

EEMBC. *Embedded Microprocessor Benchmark Consortium*. 2014. Acessado em 08 de Janeiro de 2014. Disponível em: <http://www.eembc.org>.

GAL-ON, S.; LEVY, M. *Exploring Coremark - A Benchmark Maximizing Simplicity and Efficacy*. [S.l.], 2010. Disponível em: <http://www.eembc.org/techlit/coremark-whitepaper.pdf>.

GAY, D. et al. The nesc language: A holistic approach to networked embedded systems. In: *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*. New York, NY, USA: ACM, 2003. (PLDI '03), p. 1–11. ISBN 1-58113-662-5. Disponível em: <http://doi.acm.org/10.1145/781131.781133>.

GAY D., L.-P. C. D. B. E. *NesC 1.3 - Language Reference Manual*. [S.l.], July 2009. 1 - 46 p. Disponível em: <http://nesl.ee.ucla.edu/fw/torres/home/projects/tossim-gumstix/root/nesc-1.3.1/doc/ref.pdf>.

GERWEN, J. V. V. et al. Benchmarking for wireless sensor networks. In: *The Sixth International Conference on Sensor Technologies and Applications*. Nice, France: [s.n.], 2011. (Sensorcomm, 2011).

HEMPSTEAD, M.; WELSH, M.; BROOKS, D. Tinybench: the case for a standardized benchmark suite for tinyos based wireless sensor network devices. In: *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*. [S.l.: s.n.], 2004. p. 585–586. ISSN 0742-1303.

JAIN, R. *The Art Of Computer Systems Performance Analysis:*. Wiley India Pvt. Limited, 2008. ISBN 9788126519057. Disponível em: <http://books.google.com.br/books?id=eOR0kJjgMqkC>.

KARL, H.; WILLIG, A. *Protocols and architectures for wireless sensor networks.* [S.l.]: Wiley, 2005. I-XXV, 1-497 p. ISBN 978-0-470-09510-2.

KHAN, Z. et al. Identification and analysis of performance metrics for real time operating system. In: *Emerging Technologies, 2009. ICET 2009. International Conference on*. [S.l.: s.n.], 2009. p. 183–187.

KRISHNASWAMY, U.; SCHERSON, I. A framework for computer performance evaluation using benchmark sets. *Computers, IEEE Transactions on*, v. 49, n. 12, p. 1325–1338, 2000. ISSN 0018-9340.

KURIAN, L. J. The computer engineering handbook. In: ____. [S.l.]: CRC Press LLC, 2002. cap. Performance Evaluation: Techniques, Tools and Benchmarks, p. 314 – 330.

LOUREIRO, A. A. F. et al. Wireless sensors networks (in portuguese). In: *Proceedings of the 21st Brazilian Symposium on Computer Networks (SBRC'03)*. Natal, RN, Brazil: [s.n.], 2003. p. 179–226. Tutorial.

MEMSIC. *XMesh User Manual*. McCarthy Blvd., California, EUA, 2010.

NAZHANDALI, L.; MINUTH, M.; AUSTIN, T. Sensebench: toward an accurate evaluation of sensor network processors. In: *Workload Characterization Symposium, 2005. Proceedings of the IEEE International*. [S.l.: s.n.], 2005. p. 197–203.

NYGREN, J.; CARLSSON, B. *Benchmark Simulation Model no. 1 with a Wireless Sensor Network for Monitoring and Control*. [S.l.], jan 2011.

OLIVEIRA, M. et al. Software quality metrics and their impact on embedded software. In: *Model-based Methodologies for Pervasive and Embedded Software, 2008. MOMPES 2008. 5th International Workshop on*. [S.l.: s.n.], 2008. p. 68–77.

PESOVIC, U. et al. Benchmarking performance and energy efficiency of microprocessors for wireless sensor network applications. In: *MIPRO, 2012 Proceedings of the 35th International Convention*. [S.l.: s.n.], 2012. p. 743 –747.

PETERSON, P. A. H. *From Whetstone to Benchmark Cloning - Thirty Years of Representative Synthetic Benchmarks*. [S.l.], Abril 2010. Report. Disponível em: <http://tastytronic.net/pedro/docs/synbench.pdf>.

REZAEI, Z.; MOBININEJAD, S. Energy saving in wireless sensor networks. In: *International Journal of Computer Science and Engineering Survey (IJCSES)*. [S.l.: s.n.], 2012. (1, 1), p. 23 – 37.

SPEC. *Standard Performance Evaluation Corporation (SPEC)*. 2014. Acessado em 07 de Janeiro de 2014. Disponível em: <http://spec.org>.

TINYOS. *TinyOS*. January 2013. Acessado em 23 de Janeiro de 2014. Disponível em: <www.tinyos.net>.

WEISS, A. R. *Dhrystone Benchmark - History, Analysis, Scores and Recommendations*. October 2002. White Paper. Disponível em: <http://www.eembc.org/techlit/datasheets/dhrystonewp.pdf>.

YORK, R. *Benchmarking in Context: Dhrystone*. March 2002. ARM White Paper. Disponível em: <http://www.iuma.ulpgc.es/ nunez/procesadoresILP/DhrystoneMIPS-CriticismbyARM.pdf>.

Anexo A

Linpack

Este benchmark desenvolvido em NesC, fornece uma carga de trabalho do benchmark Linpack e contagem de OPS com base na equação de solução da equação linear $2/3n^3 + 2n^2 + O(n)$, onde n é o tamanho da matriz que está sendo implementada.

Este *benchmark* foi criado com base na definição matemática relacionada ao funcionamento do *benchmark Linpack*, não foi implementada tendo como base outro código de *benchmark* semelhante.

A seguir, é apresentado o código-fonte do benchmark implementado em linguagem de programação NesC.

```
#include "SensorMessage.h"

module SendingMoteC{
        uses interface Boot;
        uses interface Leds;
        uses interface AMSend as Sender;
        uses interface SplitControl as LinkLayer;
        uses interface Timer<TMilli> as SendTimer;
}

implementation {
        message_t message;
        SensorMsg * pacote;

        event void Boot.booted(){
                call LinkLayer.start();
                call Leds.led0Toggle();
        }

        event void LinkLayer.startDone(error_t result){
                pacote = (SensorMsg *)call Sender.getPayload(&message,
                    sizeof(SensorMsg));

```

```
23                  call Leds.led1Toggle();
24                  call SendTimer.startOneShot(100);
25          }
26
27          long getTimeMillis(){
28                  return call SendTimer.getNow();
29          }
30
31          double getTime(){
32                  return ((double)getTimeMillis())/1000.0;
33          }
34
35          int n = 10;
36          double ops,flops = 0.0, mflops, tempoTotal;
37          unsigned long temp;
38
39          double startTime, endTime, runTime;
40
41          void Linpack(){
42                  float mat[10][10] = {};
43                  float v[20] = {}, transposta[10][10] = {};
44                  float m,pivo,term;
45                  float valor,lambda[10] = {};
46                  float detu=1,calc;
47                  float ml[10][10] = {}, mu[10][10] = {};
48
49                  int a,b,d,i,j,l=10,c=10,positivoNegativo,cont=0,i1,i2;
50
51                  for(i=0;i<l;i++){
52                          for(j=0;j<c;j++){
53                                  valor = (float)((float)(rand()
                                       \%100000 + 100)/100000);
54                                  positivoNegativo = rand()\%2;
55
56                                  if (valor < -1 || valor >1){
57                                          valor = 1;
58                                  }
59                                  if( positivoNegativo == 0){
60                                          mat[i][j] = valor;
61                                  }
62                                  else
63                                          mat[i][j] = -1 * valor;
64                          }
65                  }
66
67                  for(a=0;a<(l-1);a++){
68                          pivo = mat[a][a];
69                          for(b=a+1;b<l;b++){
70                                  m = mat[b][a]pivo;
71                                  lambda[cont] = m;
72                                  cont++;
73                                  for(d=0;d<c;d++){
74                                          mat[b][d] = mat[b][d] - m*mat[
```

```
                                        a][d];
75                              }
76                      }
77              }
78
79              for(i=l-1;i>=0;i--){
80                      term = mat[i][c-1];
81                      for(j=l-1;j>i;j--){
82                              term = term -v[j]*mat[i][j];
83                      }
84                      v[i]=term/ mat[i][i];
85              }
86
87              for(i=0;i<l;i++){
88                      for(j=0;j<(c-1);j++){
89                              ml[i][i] = 1;
90                      }
91              }
92
93              cont = 0;
94              for(j=0;j<(c-1);j++){
95                      for(i=j+1;i<l;i++){
96                              ml[i][j] = lambda[cont];
97                              cont++;
98                      }
99              }
100
101             for(i=0;i<l;i++){
102                     for(j=0;j<(c-1);j++){
103                             mu[i][j] = mat[i][j];
104                     }
105             }
106
107             for(i=0;i<l;i++){
108                     for(j=0;j<(c-1);j++){
109                             if(i==j){
110                                     detu *= mu[i][j];
111                             }
112                     }
113             }
114             return;
115     }
116
117     double Linpack_mflops(double *nflops, double *time){
118
119             startTime = getTime();
120             Linpack();
121             endTime = getTime();
122             runTime = endTime - startTime;
123             ops = (2.0*(n*n*n)) /3.0 + 2.0*(n*n);
124             mflops = ops/runTime;
125     ops/(1.0e6*runTime);
126
```

```
127                 *nflops = mflops;
128                 *time = runTime;
129 
130                 return runTime;
131         }
132 
133         event void SendTimer.fired(){
134                 double  time;
135 
136                 call Leds.led1Toggle();
137 
138                 Linpack_mflops(&flops,&time);\\
139                 pacote->flops = flops;
140                 pacote->time = time;
141 
142                 call Sender.send(AM_BROADCAST_ADDR, &message, sizeof(
                       SensorMsg));
143 
144                 call Leds.led2Toggle();
145         }
146 
147         event void LinkLayer.stopDone(error_t result){}
148 
149         event void Sender.sendDone(message_t *m, error_t error){
150                 call SendTimer.startOneShot(100);
151         }
152 }
```