UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ANGELITA BARBOSA NUNES

ANÁLISE DE RESULTADOS *BENCHMARK*S PARA ESCALONAMENTO DE TAREFAS DE TEMPO REAL EM MÁQUINAS HETEROGÊNEAS

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2016

ANGELITA BARBOSA NUNES

ANÁLISE DE RESULTADOS *BENCHMARK*S PARA ESCALONAMENTO DE TAREFAS DE TEMPO REAL EM MÁQUINAS HETEROGÊNEAS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Erikson Freitas Moraes
Co-orientador: Ms. Eduardo Bezerra Valentin

PONTA GROSSA

2016

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa
Diretoria de Graduação e Educação Profissional
Departamento Acadêmico de Informática
Bacharelado em Ciência da Computação

TERMO DE APROVAÇÃO

ANÁLISE DE RESULTADOS *BENCHMARK*S PARA ESCALONAMENTO DE TAREFAS DE TEMPO REAL EM MÁQUINAS HETEROGÊNEAS

por

ANGELITA BARBOSA NUNES

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 17 de novembro de 2016 como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação. A candidata foi arguida pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. Dr. Erikson Freitas Moraes
Orientador(a)

_____
Prof. Dr. Augusto Foronda
Membro titular

_____
Prof. Dr. Richard Duarte Ribeiro
Membro titular

_____
Prof. Dr. Augusto Foronda
Responsável pelo Trabalho de Conclusão de Curso

_____
Prof. Dr. Erikson Freitas de Morais
Coordenador do curso

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso -

RESUMO

NUNES, Angelita Barbosa. **Análise de resultados *benchmark*s para escalonamento de tarefas de tempo real em máquinas heterogêneas**. 2016. 34f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2016.

Para avaliar um escalonador de tarefas de tempo real em um sistema embarcado e heterogêneo é necessário gerar conjunto de tarefas que o auxilie neste processo. Este trabalho apresenta uma análise comparativa entre as seguintes ferramentas geradoras de tarefas: *Task Graph for Free* (TGFF), *Embedded System Synthesis Benchmarks Suite* (E3S), e *UUniFast*. Para realizar a comparação das ferramentas, as características foram encontradas na literatura. Assim, foram comparadas questões como tipo de tarefas geradas, customização dos algoritmos geradores, disponibilização de parametrização para gerar as tarefas, entre outros. Em seguida, os resultados encontrados são descritos, explicitando que cada ferramenta tem sua devida finalidade.

Palavras-chave: Escalonamento. Tarefas de tempo real. *Benchmark*. Sistemas heterogêneos. Sistemas embarcados.

ABSTRACT

NUNES, Angelita Barbosa. ***Benchmark*s results analysis for real-time task scheduling on heterogeneous**. 2016. 34f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Federal Technology University - Paraná. Ponta Grossa, 2016.

To evaluate a real-time task scheduler for embedded and heterogenous system, it is necessary to generate task sets for this process. This research work presents a comparative analysis among the following task generating tools: *Task Graph for Free* (TGFF), *Embedded System Synthesis Benchmarks Suite* (E3S), and *UUniFast*. To compare the tools, each tool's characteristics were found in the literature. Therefore, questions like task types, customizing of the generating algorithms, parameters to generate the tasks, among others. Finally, the findings are described, stating that each task generating tool has its own purpose.

Keywords: Scheduling. Real-time tasks. *Benchmark*. Heterogeneous systems. Embedded systems.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Sistemas computacionais foram primeiramente inseridos na comunidade militar a partir de computadores de grandes proporções e pequena capacidade computacional, mas que já realizavam operações de elevada complexidade, consideradas humanamente tediosas, que levariam centenas de anos para serem solucionadas (Stevenson, 2014). Com o tempo, esses sistemas foram apresentados à sociedade como algo acessível às pessoas comuns, não apenas por cientistas e militares (STEINBERG; SANGHERA, 2007). Os computadores proporcionavam a criação, edição, exclusão e armazenagem de documentos, planilhas e apresentações, além de cálculos matemáticos. Contudo essas poucas funcionalidades não eram suficientes para atender à crescente demanda por maior poder computacional, diversidade de aplicações e maior portabilidade. Para atender a todos esses requerimentos, pesquisadores e fabricantes investiram em grandes projetos inovadores que transformaram definitivamente a vida das pessoas. Computadores ficaram cada vez menores e sistemas computacionais passaram a integrar produtos comumente utilizados no dia a dia das pessoas através de uma tecnologia conhecida por sistemas embarcados.

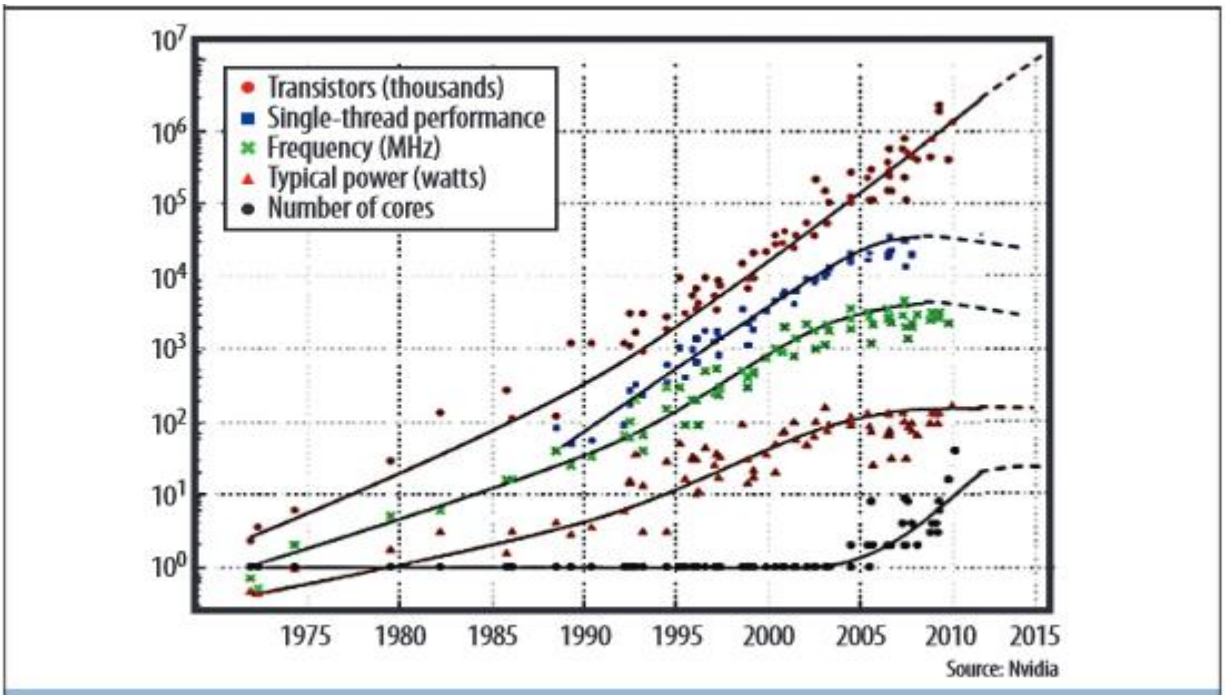
Sistemas embarcados estão cada vez mais inseridos no cotidiano das pessoas. Kothari et al (2012, p.2) afirma que sistemas embarcados são computadores programáveis. Contudo, diferentemente dos computadores pessoais, um sistema embarcado destina-se a execução de uma tarefa específica (EMBEDDED..., 2016). Como exemplos, destacam-se, sistemas de controle de acesso biométrico, controle de temperatura de ar-condicionado, MP3 players, impressoras e equipamentos de rede.

O aumento da velocidade de processamento geralmente implica no aumento da quantidade de energia que o processador consome e na quantidade de calor que ele dissipa (MIMS, 2010). Um processador mais sofisticado opera com maior frequência consumindo mais energia. Ao consumir mais energia, ocorre o aumento na temperatura, evento explicado pelo Efeito Joule. Com tanta energia sendo dissipada, está ficando cada vez mais difícil resfriar as máquinas a fim de garantir seu funcionamento adequado. Uma solução encontrada para aliviar a sobrecarga do processador foi a criação de arquiteturas

com mais de um núcleo de processamento (Kong et al, 2012). Desse modo, o processador trabalharia mais em um único ciclo de *clock*.

Entretanto, a Figura 1 mostra que o crescimento de núcleos, frequência, desempenho de uma única *thread* e consumo de energia não se mantêm constantes. Portanto, o aumento de núcleos não é uma solução que resolve por completo o problema de desempenho e consumo de energia. Uma outra solução é a computação heterogênea, que visa a integração de mais de um processador no mesmo sistema. Assim, um único sistema pode ter diferentes processadores especializados em tarefas específicas, gerando um consumo de energia mais eficiente (AMD, 2016).

Figura 1 - Evolução do consumo de energia de um processador



Fonte: Leavitt (2012)

Com mais recursos disponíveis, é necessário que haja uma distribuição justa das tarefas a serem executadas. Para isso é usado o escalonador de tarefas. Existem tarefas simples, que nem sempre exigem muito poder computacional, mas também existem as mais complexas, que algumas vezes podem exigir todo o poder computacional de uma máquina (SILBERSCHATZ, 1998). Algumas tarefas fazem o processador trabalhar

exaustivamente, obrigando-o a consumir muita energia para sua execução. O escalonador tem que lidar com todos esses tipos.

Técnicas de escalonamento geralmente priorizam desempenho de processamento mais que consumo energético (KONG et al, 2012). Contudo, Kong *et al* (2012) aponta que o escalonamento de tarefas com controle de temperatura leva em consideração a atual temperatura do processador, a temperatura máxima que ele pode atingir e ainda quanto a temperatura pode aumentar devido ao processamento de uma tarefa. A partir da definição de um modelo de estimativa térmica de tarefas e ainda a definição de um limiar de temperatura o escalonador monta a fila de prioridade. Às tarefas que tendem a não exceder o limiar de temperatura do processador, de acordo com o modelo de estimativa térmica, é atribuído alta prioridade levando em consideração o atual nível de aquecimento do processador. Por outro lado, às tarefas que tendem a exceder o limite de temperatura é atribuído baixa prioridade.

Conhecendo a importância do escalonador em um sistema computacional, é necessário atrelar suas atribuições a uma máquina capaz de lidar com as necessidades das mais variadas tarefas. Com o escalonador ciente de todos os recursos computacionais disponíveis, cabe a ele designar qual o melhor destino para as tarefas. Por sua vez, a máquina deve dar todo suporte para a execução das mesmas. Sabendo que determinadas tarefas exigem mais do processador que outras, é crucial primeiro definir se a máquina é capaz ou não de suportar tais tarefas (SILBERSCHATZ, 1998).

Uma maneira de avaliar um sistema computacional ao ponto de obter seu desempenho pode ser a partir da execução de *benchmarks*. *Benchmarks* são ferramentas que auxiliam na comparação de velocidade, desempenho de componentes de *hardware*, dentre outros aspectos, podendo ser aplicados até mesmo para obter o desempenho de um *software* em comparação a outro (CHIAPPETTA, 2012).

O uso de *benchmarks* para comparação de *hardware* e seus componentes tem se tornado uma prática cada vez mais comum, pois a partir desse resultado é possível determinar o modelo mais indicado para determinada aplicação. Fabricantes em geral veem na comparação de resultados de *benchmarks* uma oportunidade de mostrar a qualidade de seu produto. Assim, é possível acompanhar o desenvolvimento de novos processadores que oferecem benefícios como alto desempenho e baixo consumo de

energia e ainda verificar a qualidade desse produto em comparação a outros existentes no mercado.

Este capítulo tem como finalidade apresentar o problema de pesquisa e seu contexto. Além disso, as motivações para realizá-lo também são descritas. Da mesma forma, são apresentados os objetivos que nortearão a execução da pesquisa, e, por fim, uma descrição estrutural da organização do conteúdo elaborado.

1.1 DESCRIÇÃO DO PROBLEMA

Existem no mercado *benchmarks* para os mais diversos fins. Há aqueles que avaliam o desempenho de processadores na execução de diferentes tarefas, e ainda outros que avaliam o consumo energético do processador. Existem também *benchmark*s que avaliam sistemas de tempo real. Há ainda opções de *benchmarks* que avaliam o escalonador através da geração de tarefas de tempo real. Esse trabalho visa comparar três benchmarks para geração de tarefas de tempo real e assim sugerir ao pesquisador as opções mais indicadas de uso desses *benchmarks*.

1.2 MOTIVAÇÃO

Sistemas embarcados estão inseridos de tal modo à sociedade que certamente a extinção deles afetaria radicalmente o mundo. A busca constante do homem em superar suas próprias invenções e garantir à sua espécie maior comodidade e praticidade no dia a dia, fez com que houvessem muitos avanços computacionais. E um desses avanços foi a inserção de tarefas mais complexas, como tarefas de tempo real. Acarretando em pesquisas sobre escalonamento de tais tarefas (SILBERSCHATZ, 1998).

Profissionais que trabalham com escalonamento de tarefas de tempo real em sistemas embarcados podem encontrar no resultado dessa pesquisa um respaldo para suas próprias pesquisas. Eles saberiam qual o benchmark mais adequado para cada situação.

1.3 OBJETIVOS

Os objetivos são subdivididos em duas categorias: o objetivo geral do trabalho e os objetivos específicos. No objetivo geral é apresentada qual a finalidade do projeto, qual resultado espera-se obter a partir dele. Por outro lado, os objetivos específicos apresentam metas que precisam ser atingidas para atingir o objetivo geral.

1.3.1 Objetivo geral

Este trabalho tem como objetivo principal realizar uma comparação de três diferentes *benchmark*s destinados a avaliar escalonadores de tarefas de tempo real em sistemas embarcados a partir da geração de conjuntos de tarefas.

1.3.2 Objetivos específicos

Os objetivos específicos que auxiliam na obtenção do resultado final são os seguintes:

- Pesquisar por benchmarks geradores de tarefas para sistemas de tempo real;
- Entender como funciona um *benchmark* e quais são os seus passos para avaliação de um escalonador de tarefa de tempo real;
- Buscar as características de cada *benchmark*;
- Comparar as características dos *benchmarks* entre si.

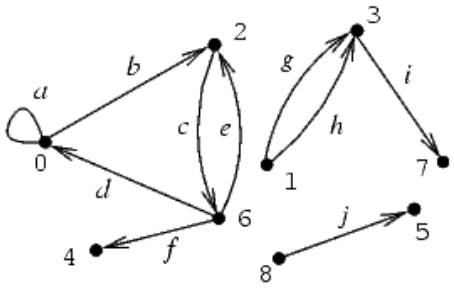
2 CONCEITOS E DEFINIÇÕES

Para o desenvolvimento dessa pesquisa serão utilizados os conceitos de grafos, sistema de tempo real, processadores heterogêneos em sistemas embarcados, *benchmark* e consumo de energia. Feofiloff (2015), Loureiro (2015) e Mariani destacam as definições gerais de grafos. Os autores Laplante (2006), Farines et al (2000) e Valentin (2010) em seus trabalhos explicam o conceito de sistema de tempo real e suas aplicações. Por outro lado, Uchiyama et al (2012) descreve os processadores heterogêneos em sistemas de tempo real.

2.1 GRAFOS

De modo geral, um grafo é uma representação de relações entre objetos e seus pares (FEOFILOFF, 2015). Em outras palavras, ele é composto por um conjunto de vértices e um conjunto de arestas que conectam esses vértices. Vértices, também chamados de nodos ou nós, são os elementos, na Figura 2, os vértices são representados pelos números *0, 1, 2, 3, 4, 5, 6, 7 e 8*. Os conectores entre os vértices são chamados de arestas, caminhos ou arcos, que na Figura 2 são representados pelas letras *a, b, c, d, e, f, g, h, i* e *j*. As arestas podem ser de diferentes tipos, orientadas (quando há ordem de origem e destino), não-orientadas (não há ordem de vértices). Desta forma, um grafo onde todas as arestas são orientadas é chamado de grafo orientado ou dígrafo. Logo, quando não há orientação de arestas, o grafo é chamado de não-orientado (MARIANI, 2016). Os grafos da Figura 2 são orientados, cada vértice pode ser a origem ou o destino de uma aresta, por exemplo o vértice *0* é a origem da aresta *b* que conduz ao vértice de destino *2*. Assim formam-se os caminhos.

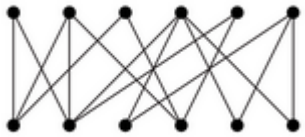
Figura 2 - Exemplos de grafos



Fonte: Feofiloff, 2015

Caminho é a sequência entre um vértice e outro. Desta forma, o vértice escolhido para iniciar o caminho é chamado de vértice inicial e o final é chamado vértice terminal. Um ciclo ocorre quando o mesmo vértice é o início e o final de um caminho (LOUREIRO, 2015), na Figura 2 é possível encontrar um ciclo que se inicia em *0*, a partir da aresta *b* vai para o vértice *2*, de onde sai a aresta *c* que leva ao vértice *6*, onde é possível escolher a aresta *d* que leva de volta ao vértice *0*. Quando não há ciclos em um grafo, ele é chamado de acíclico (FEOFILOFF, 2015). Feofiloff (2016) define ainda que um grafo também pode ser bipartido, significando que é possível repartir o conjunto de vértices de um grafo em dois subconjuntos de forma que cada ponta de uma aresta pertença a um subconjunto diferente, como é possível ver na Figura 3.

Figura 3 - Exemplo de grafo bipartido



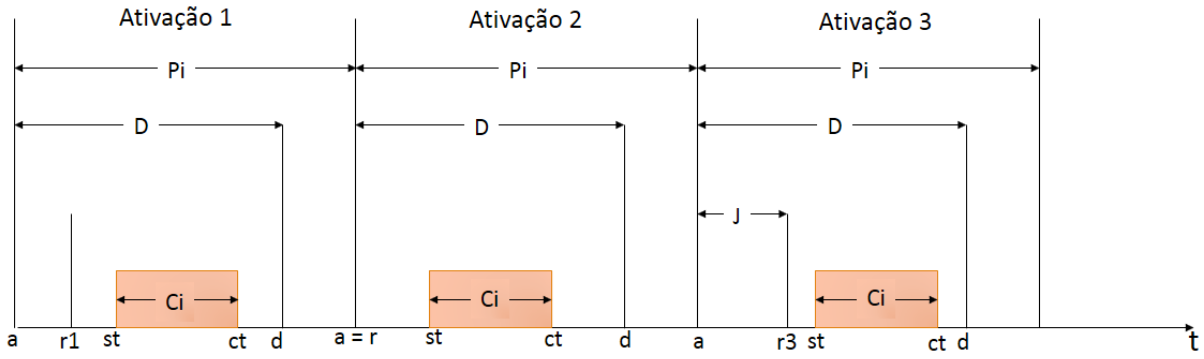
Fonte: Feofiloff, 2016.

2.2 SISTEMAS DE TEMPO REAL

Um sistema de tempo real é aquele que precisa executar suas tarefas em um determinado intervalo de tempo e com maior acurácia possível (SILBERSCHATZ, 1998).

Contudo, ainda de acordo com Silberschatz (1998), esses sistemas ainda podem variar de acordo com a real necessidade de que os resultados sejam pontuais e precisos. Neste caso, cada sistema pode apresentar complexidades distintas. De acordo com Farines (2000), “um Sistema de Tempo Real deve ser capaz de oferecer garantias de correção temporal para o fornecimento de todos os seus serviços que apresentem restrições temporais”

Figura 4 - Tarefas de tempo real



Fonte: Farines et al, 2000

A ilustração da Figura 4 demonstra a ativação de tarefas periódicas a partir dos tempos de chegada (a_i), de liberação (r_i), de início (st_i), término (ct_i), e os *deadlines* absolutos (d_i). Sendo *i* a representação de uma sequência que se inicia com *i* = 1, tem-se que a Ativação 1, que possui período $P_{i=1}$, inicia em $a_{i=1}$, que é o “tempo em que o escalonador toma conhecimento de uma ativação dessa tarefa” (FARINES et al, 2000, p. 13). A liberação da tarefa inicia-se a partir de $r_{i=1}$, entretanto, a tarefa só começa a ser computada em $st_{i=1}$. Por sua vez, $ct_{i=1}$, representa o tempo final da computação da tarefa. A partir do tempo inicial e do tempo final de computação é possível saber o tempo total de computação de uma tarefa, *C*. Já $d_{i=1}$ representa o tempo máximo de computação de uma tarefa, o ideal é que esse tempo seja sempre respeitado. Outro elemento encontrado na Figura 4 é o *J*, chamado *jitter*, que acontece quando há variância entre o tempo de chegada de uma ativação e o tempo de liberação da tarefa.

A complexidade e a importância de cada sistema, ao ponto que um simples erro, por menor que seja, possa gerar danos irreparáveis, ocasionou na categorização dos

sistemas de tempo real. De acordo com Farines et al (2000), existem duas categorias: sistemas críticos e não críticos de tempo real, as quais serão descritas a seguir.

Se uma falha ocasionada pelo tempo em um sistema de tempo real acarreta em danos consideravelmente superiores aos ganhos de um sistema operando em condições normais, então esse sistema de tempo real é chamado crítico (Farines et al, 2000). Por exemplo, um sistema de medição de sensores de um reator em uma usina nuclear de energia precisa ser muito preciso em suas medidas e também no tempo de resposta de cada uma. A falha no tempo de resposta nesse caso, pode ocasionar uma grande tragédia.

Por outro lado, quando as consequências da falha de um sistema de tempo real produzem resultados não tão distantes que os ganhos em sistemas operando em condições normais, esse sistema é denominado sistema de tempo real não crítico (FARINES et al, 2000). Um exemplo, o *player* de áudio mp3, que além de processar corretamente cada descompressão de amostras de áudio, precisa tocá-las em um tempo que não cause problemas no áudio, como *delay*.

2.2.1 Escalonamento de Tarefas de Tempo Real

Uma tarefa de tempo real possui restrições temporais e seu resultado deve ser extremamente preciso (Valentin 2010). Portanto, esse tipo de tarefa precisa cumprir um determinado prazo, ou uma meta temporal. Além disso, tarefas disputam entre si os recursos do processador.

De acordo com Valentin (2010, p. 9), é preciso deixar claro os conceitos de escala, escalonador e escalonamento antes de discutir sobre escalonamento de tempo real. Primeiramente, “escala é uma lista ordenada que indica a ordem de ocupação do processador pelas tarefas ‘prontas’”. Segundo, o escalonador é o ator que determina qual tarefa pronta irá utilizar o processador para executar suas operações. E, por fim, escalonamento é a regra aplicada para ordenar as tarefas que irão utilizar o processador. Portanto, escalonamento de tempo real é o conjunto de regras aplicadas para escalonar tarefas de tempo real respeitando suas restrições temporais (Farines et al, 2000).

2.2.2 Algoritmos de Escalonamento

Os algoritmos de escalonamento são classificados como estáticos ou dinâmicos, como aponta Valentin (2010). Na formação da escala, os algoritmos estáticos são aqueles cujos parâmetros para a determinação da lista são fixos. Em contrapartida, quando esses mesmos parâmetros podem variar de acordo com o progresso do sistema, o algoritmo é dinâmico. O que faz um algoritmo de escalonamento ótimo é ele conseguir analisar a classe problema e a partir dela definir a melhor maneira de resolve-la, seja maximizando ou minimizando alguma parte do processo (Valentin, 2010).

2.3 PROCESSADORES HETEROGÊNEOS E EMBARCADOS

Processadores heterogêneos são caracterizados pelo desenvolvimento de processamento paralelo dentro do seu *board* com o intuito de aumentar desempenho e diminuir o consumo de energia. Isso significa que, um processador heterogêneo tem, em sua arquitetura, diversos outros tipos de processadores com funções especificas como processamento de sinais digitais, de mídia, entre outros (Uchiyama et al 2012).

Em sistemas embarcados, processadores heterogêneos são largamente utilizados para obter ganho de desempenho e eficiência energética. Por exemplo, alguns processadores heterogêneos para sistemas embarcados disponíveis no mercado, como os *Snapdragons*, que possui alguns processadores de propósito geral de alto desempenho e alto consumo de energia, e também possui processadores de propósito geral com baixo consumo de energia, mas com baixo desempenho. Alguns componentes do *Snapdragons* são, *Central Processing Unit* (CPU), *Digital Signal Processing* (DSP), *Graphics Processing Unit* (GPU), entre outros para lidar com funções de *smartphones* como câmeras, vídeo e display.

2.4 BENCHMARK

Benchmark pode ser definido como o conjunto dos resultados de indicadores de medidas de desempenho de um processador a partir da execução de determinadas operações (CHIAPPETTA. 2012). Esses indicadores normalmente são quantificáveis com medidas de tempo de execução. Chiappetta (2012) afirma que, dentre os vários tipos de *benchmarks*, existem aqueles que realizam testes sintéticos, os quais não refletem problemas do mundo real, enquanto outros baseiam-se inteiramente em aplicações que simulam cargas de trabalho reais. O autor diz ainda que há *benchmarks* que avaliam a eficiência de apenas alguns componentes, porém há aqueles que avaliam o sistema todo. Em geral, os *benchmarks* estressam um ou mais componentes, o máximo possível, com excessiva carga de trabalho, para garantir que tais componentes estão trabalhando corretamente (CHIAPPETTA. 2012).

O desempenho de um sistema computacional depende do trabalho conjunto de diversos componentes, como “CPU, memória, placas gráficas e configurações de armazenamento” (CHIAPPETTA. 2012). Porém, sistemas computacionais são utilizados para os mais diversos fins. Uma pessoa que utiliza um computador para jogar tem mais interesse em saber o desempenho da placa gráfica do que da memória, por exemplo. Existe ainda os *benchmarks* que avaliam *softwares*, como escalonadores.

De acordo com Jelenković et al. (2009), a eficiência de um escalonamento é medida a partir da capacidade do sistema em completar o maior número de tarefas no menor tempo possível usando poucos recursos do sistema. Os autores afirmam que para avaliar a eficiência de um escalonador é necessário um conjunto predefinido de tarefas. Esse conjunto pode ser de tarefas dependentes umas das outras ou de tarefas completamente independentes. É possível avaliar vários aspectos do escalonador a partir dos conjuntos de tarefas, por exemplo, como o escalonador se comporta frente a um sistema multiprocessado, a tarefas com diferentes prioridades e requisitos (JELENKOVIĆ et al. 2009).

3 TRABALHOS RELACIONADOS

Esse capítulo será dividido entre duas frentes de trabalhos relacionados. A primeira refere-se a trabalhos na área de escalonamento de tarefas de tempo real em sistemas heterogêneos de processamento. Já a segunda parte refere-se a pesquisas com *benchmark*s.

3.1 ESCALONAMENTO DE TAREFAS DE TEMPO REAL EM SISTEMAS HETEROGÊNEOS

Em um sistema embarcado com processamento heterogêneo o escalonamento de tarefas deve ser feito de modo que cada tarefa seja destinada ao processador que além de ser o ideal para aquela determinada tarefa, ainda consuma a menor quantidade de energia possível. Após observar que ocorriam variações no tempo de execução e no consumo de energia em diferentes aplicações, Alahmad e Gopalakrishnan (2011) desenvolveram um novo algoritmo de distribuição de tarefas entre os processadores de maneira eficiente levando em consideração restrições de tempo e consumo de energia, sem afetar a qualidade de serviços.

Alahmad e Gopalakrishnan (2011) determinaram a relação entre cada processador, sua velocidade, tarefas e processadores levando em consideração alguns requisitos, tais como: a velocidade do processador deveria ser constante durante todo o processamento; o consumo total de energia deveria ser o mínimo possível; o processador não pode exceder sua capacidade; todo o processamento de uma tarefa deveria ser feito em um único processador. Ao atingir todos os requisitos, os autores implementaram simulações em plataformas de médio e pequeno porte para certificar a qualidade da solução final. Eles utilizaram programação dinâmica para chegar a um algoritmo quase ótimo em tempo polinomial.

O escalonamento de tarefas é um tema bastante visado em sistemas operacionais, uma vez que é papel do sistema operacional gerenciar os recursos da máquina (FARINES et al, 2000). Contudo, a distribuição de tarefas entre processadores heterogêneos respeitando os *deadlines* de cada tarefa e ainda levando em consideração

o consumo de energia é um problema muito maior e bastante visado na área da computação. Chen et al. (2011) atesta que o problema se torna ainda maior quando uma mesma tarefa pode ser dividida entre diferentes processadores. Os autores estudaram o problema de alocar cada tarefa a um processador específico sem extrapolar a capacidade computacional de cada processador e respeitando o *deadline* de cada tarefa. A heurística utilizada pelos autores é a da otimização da colônia de formigas (DORIGO, 1997). Essa heurística é baseada no comportamento das formigas que sempre utilizam o menor caminho entre sua casa e a fonte de comida. A maior vantagem dessa heurística sobre as outras é o fato dela se adaptar as mudanças constantes da instância do problema. Os autores afirmam que o algoritmo deles foi desenvolvido para otimizar o uso dos recursos disponíveis e o consumo de energia.

A metodologia utilizada foi a representação do problema de atribuição de tarefas em um grafo bipartido com duas categorias de nós, são elas: conjunto de tarefas periódicas e plataforma de processamento heterogêneo com *n* processadores. De acordo com Chen et al. (2011) uma tarefa é mapeada no conjunto de tarefas periódicas e um processador é mapeado no conjunto de processadores de uma plataforma heterogênea de multiprocessamento. Só existirá uma aresta entre os dois conjuntos se, e somente se, a tarefa puder ser alocada àquele processador sem que a capacidade computacional dele seja violada. Os autores afirmam ainda, que o custo da atribuição está diretamente relacionado ao consumo de energia que será utilizado para completar a tarefa. A partir de um conjunto de tarefas e um conjunto de processadores heterogêneos, cada tarefa é incumbida a um processador até que todas as tarefas sejam alocadas ou até que nenhuma das tarefas restantes possa ser alocada a um processador sem extrapolar sua capacidade computacional. Se a distribuição de tarefas para com pelo menos uma tarefa sem um processador atribuído a ela, então o resultado final se mostrava não factível. Contudo, se todas as tarefas são devidamente alocadas, então o resultado final apresentava uma solução viável. Portanto, o resultado é parcial se uma ou mais tarefas não forem alocadas.

Finalmente, os autores notaram que sua abordagem solucionava não apenas o problema de alocação de tarefas a processadores. Ela auxiliava também na busca por tarefas executáveis e com baixo consumo de energia.

Para ampliar a capacidade computacional de dispositivos, melhorar a qualidade das aplicações e ainda diminuir o consumo de energia, uma das soluções encontradas foi a implantação de múltiplos processadores em um sistema (SILBERSCHATZ, 1998). A dificuldade está em particionar as tarefas de tempo real entre os processadores de modo a minimizar consumo de energia e sem sobrecarregar os processadores.

Os autores Chen e Thiele (2008) apresentaram um *framework* que realiza distribuição de tarefas periódicas de tempo real em um sistema com dois tipos de processadores. Através de experimentos, os pesquisadores provaram ainda que seu *framework* é quase ótimo na minimização do consumo de energia.

Chen e Thiele (2008) utilizaram tarefas periódicas de tempo real que não possuem dependências entre si. Os autores pesquisaram meios de minimizar o consumo de energia de tais tarefas em sistemas com dois processadores. O objetivo de Chen e Thiele era de dividir um conjunto *n* de tarefas em dois grupos, T1 e T2, de modo que o grupo de tarefas T1 seria executado em um processador e o conjunto T2 no outro, ambos sendo escalonados pelo algoritmo de escalonamento de tarefas *Ealiest-Deadline First* (EDF), que prioriza as tarefas de acordo com seus *deadlines*. Por exemplo, quando mais cedo a deadline, mais prioritário se torna a tarefa. (SILBERSCHATZ, 1998).

Os experimentos realizados por Chen e Thiele mostraram que os objetivos iniciais foram atingidos com sucesso. O deadline das tarefas foi devidamente respeitado e a minimização do consumo de energia, mesmo no pior caso, foi garantida.

A economia nas contas de energia elétrica e a maximização do tempo de vida da bateria de um sistema embarcado foram fatores fundamentais que motivaram a pesquisa de Chen e Thiele (2011). Diante de uma variedade de unidades de processamento, cada qual com seu distinto modelo de consumo de energia, os autores aplicaram a estratégia de sempre selecionar primeiro o processador com o menor consumo médio de energia em seu algoritmo de escalonamento de tarefas. Os autores se apoiam no modelo de consumo médio de cada unidade de processamento proposto por trabalhos correlatos.

Sabendo previamente o consumo médio de energia dos processadores e sua capacidade computacional, as tarefas alocadas devem respeitar os limites de processamento do processador (CHEN; THIELE, 2011). Para tanto, dada uma entrada de *n* tarefas e *m* unidades de processamento, o objetivo definido por Chen e Thiele (2011)

é que o algoritmo tenha como resultado a distribuição de todas as tarefas entre as unidades de processamento respeitando a capacidade computacional de cada processador e o tempo de execução de cada tarefa. Essa distribuição é feita selecionando sempre o processador com menor consumo médio de energia. Vale ressaltar que a tarefa é alocada ao processador que consegue executar o maior número de tarefas com o menor consumo de energia.

O Quadro 1 mostra resumidamente os temas abordados nos trabalhos relacionados. É possível notar que todos eles abordam os temas: sistemas de tempo real, multiprocessamento e escalonamento. Apenas dois falam sobre sistemas heterogêneos.

Quadro 1 - Comparação dos assuntos abordados pelos autores citados na seção 3.1

Autor	Tempo Real	Multiprocessamento	Escalonamento	Sistemas Heterogêneos
Alahmad e Gopalakrishnan (2011)	X	X	X	X
Chen et al. (2011)	X	X	X	X
Chen e Thiele (2008)	X	X	X	
Chen e Thiele (2011)	X	X	X	

Fonte: Autoria própria

3.2 BENCHMARKS

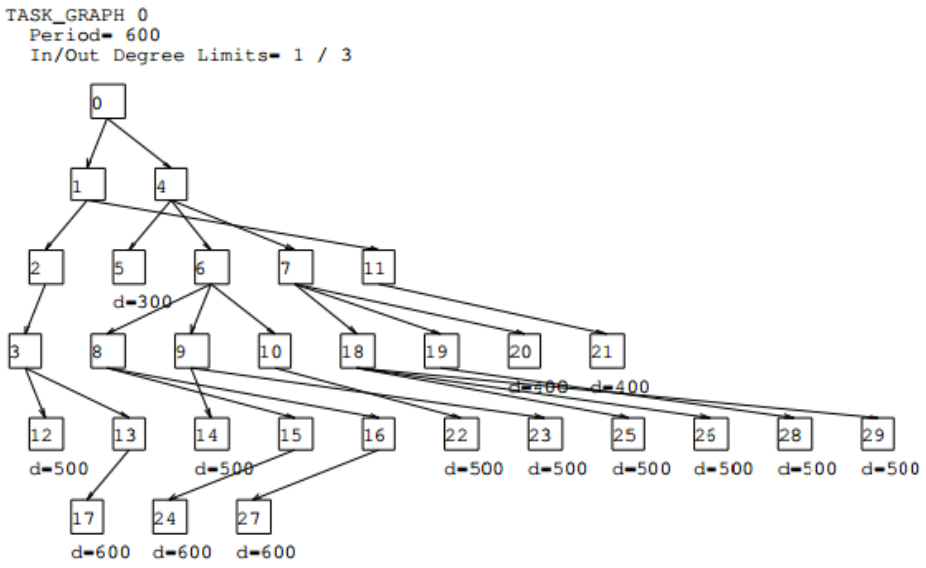
Os *benchmarks Embedded System synthesis benchmarks suite* (DICK, 2008), *Task graphs for free* (DICK et al. 1998) e *UUniFast* (BINI; BUTAZZO, 2004) são fundamentais para esta pesquisa por avaliarem a atuação dos escalonadores de tarefa de tempo real.

3.2.1 *Task Graphs for Free* (TGFF)

Dick et al (1998) desenvolveram o TGFF, o qual consiste em um gerador pseudorrandômico de tarefas em formato de grafos acíclicos, manipulável pelo usuário, que cria tarefas para uso em pesquisas de alocação e escalonamento de tarefas em sistemas embarcados e multiprocessados. Os autores afirmam ainda que o algoritmo gera tarefas independentes ou conjunto de tarefas que são compostas por grafos parcialmente ordenados. Durante sua execução, é gerada a descrição completa do grafo de tarefas, contendo os atributos para o processador, recursos de comunicação, tarefas, e comunicação entre as tarefas. Além disso, o operador pode modificar os parâmetros para controlar as correlações entre os atributos.

Os autores descrevem que cada nodo do grafo pode ser uma tarefa ou um processador e as arestas são as conexões de comunicação entre as tarefas/processadores. O TGFF suporta otimização de tempo de execução, consumo de energia, testes, e custo permitindo um número arbitrário de atributos os quais podem ser correlacionados ou não, para ser associados a cada processador e recurso de comunicação.

Figura 5 - Exemplo de um grafo gerado pelo TGFF.



Fonte: Dick et al (1998)

Os requerimentos para a implementação do algoritmo TGFF são mínimos, uma vez que ele necessita apenas de um compilador de linguagem C++ e ainda uma ferramenta adicional, VCG, que favorece a visualização dos grafos. Outra grande vantagem do TGFF é a possibilidade de visualizar os grafos gerados através de imagens. A Figura 5 mostra a representação gráfica de um grafo, onde é possível que ele possui período, *deadline*, e algumas informações sobre a quantidade de arestas que saem ou chegam de um nó, uma aresta pode chegar a um nó e três podem sair.

O TGFF propicia a geração de grande variedade de tipos de grafos através de parâmetros informados diretamente pelo usuário. Outro benefício da utilização desse algoritmo é a possibilidade do próprio usuário informar a quantidade de grafos a serem fabricados, o tipo dos grafos e ainda a quantidade de nós e arestas. Além disso, uma funcionalidade extra é a geração de tabelas para maior detalhamento dos nós e arestas.

Uma das desvantagens do TGFF, de acordo com os autores, se aplica a usuários do sistema operacional *Windows*, que necessitam adaptar o arquivo *makefile* para que o *benchmark* funcione. Outra desvantagem é o fato do usuário ficar preso a duas versões do *software* uma vez que a versão antiga propicia melhores resultados na obtenção de árvores e outros tipos de grafos.

3.2.2 *Embedded System Synthesis Benchmarks Suite* (E3S)

Embedded System Synthesis Benchmarks Suite (E3S), é formado por um conjunto de *benchmark*s destinados a avaliar sistemas embarcados durante a execução de alocação, atribuição e escalonamento de tarefas desenvolvido por Dick (2008). A execução do *benchmark* se dá a partir do escalonamento das tarefas já previamente geradas. O E3S usa o TGFF, portanto as tarefas e os processadores são organizados em grafos. Dentre os arquivos providos pelo E3S estão informações sobre os processadores, sobre as tarefas e o sistema de comunicação entre as tarefas/processadores.

O *benchmark* apresenta um catalogo de tarefas destinados a avaliar sistemas embarcados, especificamente sistemas voltados para as seguintes áreas: industrial/automobilística, consumidor, redes, escritório e telecomunicações (DICK, 2008). Além das tarefas, o benchmark traz também um conjunto de processadores que

se relacionam com as tarefas. Todas as informações necessárias para a geração dos grafos já vêm prontas e organizadas, assim o usuário pode realizar os testes, sem se preocupar com as tarefas.

3.2.3 UUnifast

Chen e Thiele (2008), afirmam que um meio eficaz de avaliar o desempenho de um teste de escalonamento de tarefas de tempo real em sistemas embarcados é através da geração de um grande número tarefas randômicas e da verificação da percentagem de tarefas realizadas com sucesso. O algoritmo de geração de tarefas randômicas UUniFast contribui através da geração de tarefas de tempo real independentes com períodos e utilização uniformemente distribuídos (BINI; BUTTAZZO, 2004). A utilização é o inverso da porcentagem de deadlines que o sistema pode perder, sendo 1 o valor máximo. Assim, se o usuário informar o valor 1 como parâmetro de utilização, então o UUniFast ao gerar uma tarefa, deve garantir que o tempo de computação será terminado antes do deadline.

De acordo com os autores Bini e Buttazzo (2004), a partir dos parâmetros, número de tarefas e porcentagem de utilização de tarefa, informados pelo usuário, o UUniFast gera vários conjuntos de tarefas que atendam aos requisitos, porém ele seleciona o conjunto mais próximo do esperado para apresentar ao usuário. Em geral, as tarefas possuem um conjunto de operações que já vêm prontas para serem executadas. As tarefas são de tempo real, portanto, possuem todas as informações necessárias para avaliar o escalonador.

O Quadro 2 apresenta uma breve comparação entre os principais temas abordados pelos benchmarks, sendo eles: geração randômica de tarefas, dependência ou independência entre tarefas, se a tarefas geradas são de tempo real, se é possível customizar uma tarefa, se o benchmark pode ou não ser utilizado em sistemas embarcados e/ou multiprocessados, se o usuário pode ou não informar os parâmetros e se há relacionamento entre tarefas e processadores. A partir dessas informações é possível realizar uma análise comparativa mais aprofundada entre os benchmarks, tal análise será vista no capítulo a seguir.

Quadro 2 – Comparação entre benchmarks.

	TGFF	E3S	UUniFast
Gerador Randômico	x		x
Tarefas independentes	x		x
Interdependência de tarefas	x	x	
Tarefas de tempo real	x	x	x
Sistemas multiprocessados	x	x	
Sistemas embarcados	x	x	x
Relaciona tarefas a processadores	x	x	
Parâmetros para criação de tarefas	x		x
Tarefas customizáveis	x		
Pronto para uso		x	x

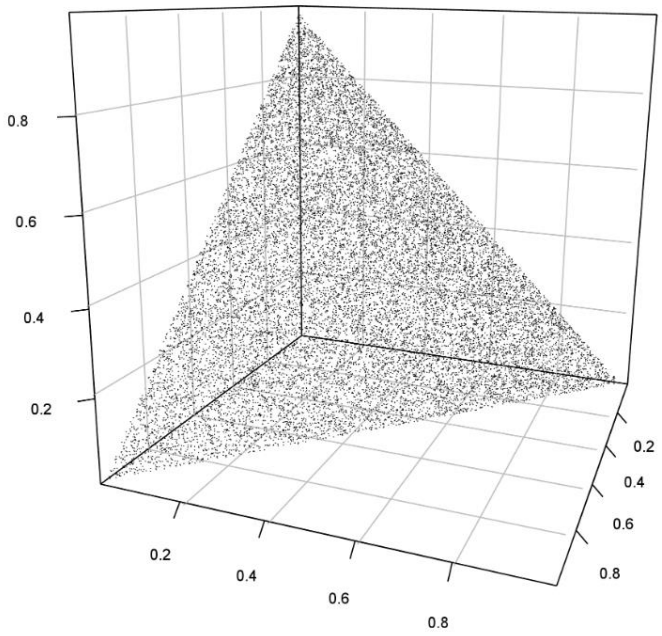
Fonte: Autoria própria

4 ANÁLISE COMPARATIVA

Os benchmarks UUnifast, TGFF e E3S são ferramentas *benchmark* utilizadas em análises de escalonadores de tarefas de tempo real. O que todos eles têm em comum é que eles geram ou fornecem conjuntos de tarefas de tempo real e são utilizados em pesquisas com sistemas embarcados, fatores que os levaram a serem escolhidos para realização dessa análise comparativa. Entretanto, eles se diferem entre si em alguns aspectos, tais como: enquanto o TGFF e o UUniFast são geradores randômicos, o E3S já apresenta conjuntos de tarefas prontos; por outro lado, o E3S e o TGFF apresentam tarefas com interdependência entre si, enquanto o UUniFast e o TGFF geram tarefas independentes; enquanto todos eles trabalham com sistemas embarcados e geram tarefas de tempo real, apenas o UUniFast não trabalha com sistemas multiprocessados e por isso não realiza o relacionamento entre tarefas e processadores; pelo fato do E3S já conter conjuntos de tarefas prontos, ele é o único que não precisa de parâmetros informados pelo usuário; contudo, o TGFF é o único cujas tarefas são customizáveis; por fim, o E3S e o UUniFast já vêm prontos para serem utilizados, ao contrário do TGFF que exige algumas informações para que suas tarefas sejam geradas. As semelhanças e as diferenças entre esses três benchmarks serão vistas e detalhadas a seguir.

O algoritmo UUniFast foi construído para gerar randomicamente conjuntos de tarefas com maior precisão na distribuição uniforme de período e utilização de cada tarefa (DI NATALE *et al*, 2013). A utilização de uma tarefa é dada pelo tempo de computação sobre o seu período. O tempo de computação é inversamente proporcional a perda de *deadline* de uma tarefa. Desta forma, quanto menor a perda de deadline, maior a utilização de uma tarefa (BINI; BUTTAZZO, 2004). Portanto, o algoritmo calcula se os valores de utilização do conjunto de tarefas atendem o valor requisitado pelo usuário. Assim, são mantidos somente os conjuntos de tarefas com o valor correto de utilização definido por parâmetro (EMBERSON *et al*, 2010). Além disso, as tarefas geradas não têm dependência entre si na ordem de execução (BINI; BUTTAZZO, 2004). A Figura 6 mostra 20.000 conjuntos de tarefas contendo 3 tarefas cada, geradas pelo UUniFast, todas com um total de utilização de 0.98, ou seja, com possibilidade de perda de deadline de 0.02.

Figura 6 – 2x10^4 conjuntos de tarefas com utilização total de 0.98.



Fonte: Emberson et al, 2010

No entanto, Emberson *et al* (2010) sustentam que o UUniFast não é muito utilizado em sistemas multiprocessados. Pesquisadores acadêmicos reconhecem que o algoritmo não consegue garantir a distribuição uniforme em sistemas multiprocessados sozinho, pois o valor de utilização total do conjunto de tarefas pode conter valor inválido de utilização individual de uma tarefa. Para contornar esse problema, os autores sugerem o uso do algoritmo Stafford's Randfixedsum para gerar um número de tarefas e valor de utilização válidos (Emberson *et al*, 2010), que não tem relevância no contexto desta pesquisa.

O TGFF, por sua vez, tem como foco a geração randômica de grafos direcionados e acíclicos de tarefas a partir de parâmetros definidos pelo usuário. Ou seja, diferentemente do UUniFast, as tarefas geradas pelo TGFF podem ou não ter dependência entre si. Como no benchmark anterior, o usuário deve, obrigatoriamente, definir o número mínimo de tarefas dos grafos. Porém, diferente do UUnifast, não existe uma preocupação com a utilização do conjunto de tarefas. Além disso, ao contrário do UUniFast que atribui um período para cada tarefa, o TGFF atribui um período para cada grafo, um conjunto de tarefas. Além disso, para cada nó terminal é atribuído um *deadline*

que deve ser respeitado a partir do nó de origem até ele. Desta forma, o sistema multiprocessado irá executar paralelamente o conjunto de tarefas de um grafo respeitando suas dependências dentro do período do grafo.

A Figura 5 mostra a representação gráfica de um grafo de tarefas cujo nome é Task_Graph 0. O período desse grafo equivale a 600 microssegundos. As tarefas recebem um tipo, representação numérica de 0 a 29. As arestas indicam as dependências entre as tarefas. Já o grau do grafo varia entre 1 e 3, portanto, de uma tarefa podem sair 1, 2 ou 3 arestas. As tarefas terminais recebem um deadline, por exemplo, a sequência que sai da tarefa de origem do tipo 0 e chega até a tarefa terminal do tipo 12 tem deadline igual a 500 microssegundos.

Cada tarefa ou aresta possui um tipo, definido ou não pelo usuário. Além dos parâmetros customizáveis, caso o usuário não queira utilizar os tipos definidos diretamente pelo TGFF, ele pode usar uma tabela para indexar tipos de tarefa/aresta. É possível observar na Figura 7 um exemplo de tabelas de parâmetros para cada tipo de tarefa. Portanto, uma tarefa do tipo 0 no Processador 0, levará 291428 microssegundos para ser completada e precisará de 348KB de memória. Observa-se também que o processador possui parâmetros próprios, tais como custo, em dólares, consumo energético em watts e área, em milímetros quadrados.

Figura 7 - Parâmetros para cada tipo de tarefa

```
PROCESSADOR 0 {
# preco (USD)       força (W)                 area (mm2)
      14            0.533                     7.5
# tipo              tempo de execução (ms)    memoria (KB)
0                   291428                    348
1                   36265                     256
2                   19323                     312
3                   43251                     724
}

PROCESSADOR 1 {
# preco (USD)       força (W)                 area (mm2)
      11            0.648                     6.9
# tipo              tempo de execução (ms)    memoria (KB)
0                   260032                    380
1                   36463                     212
2                   38269                     340
3                   26035                     792
}
```

Fonte: Dick et al (1998).

A suíte de Benchmarks para sistemas embarcados E3S (Embedded System Synthesis Benchmarks Suite) utiliza-se do TGFF para gerar grafos de tarefas. Contudo, o E3S se difere do TGFF por ser um conjunto de ferramentas projetadas para o uso na alocação, atribuição e escalonamento de tarefas (Dick, 2008). Seguindo a organização da EEMBC (Embedded Microprocessor Benchmark Consortium), a suíte conta com um conjunto de tarefas para 5 aplicações diferentes: industrial/automobilística, consumidor, redes, escritório e telecomunicações (DICK, 2008). São mais de 45 tipos de tarefas pertinentes as 5 aplicações. Os algoritmos geradores dos grafos de tarefas se baseiam nas informações de custo, tempo de execução de operações, entre outras informações, de 17 processadores diferentes do mercado, tais como AMD ElanSC520, Analog Devices 21065L, Motorola MPC555 e Texas Instruments TMS320C6203. O detalhamento completo de tarefas para cada aplicação pode ser encontrado no Quadro 3.

Quadro 3 – Conjunto de tarefas de cada aplicação do E3S

Industrial	**Telecomunicações**	**Genérica**
Conversão de tempo para ângulo	Auto correlação - Dados1 (pulso)	Tarefa para reserva de espaço
Ponto flutuante básico	Auto correlação - Dados2 (sinal)	
Manipulação de bits	Auto correlação - Dados3 (fala)	
Cache Buster	Codificador Convolucional - Dados1 (xk5r2dt)	
CAN Requisição remota de dados	Codificador Convolucional - Dados2 (xk4r2dt)	
Transformação Fast Fourier (Auto/Indust. Versão)	Codificador Convolucional - Dados3 (xk3r2dt)	
Filtro de resposta de impulso finito (Auto/Indust. Vers)	Atribuição de bits de ponto fixo - Dados2 (typ)	
Filtro de resposta de impulso infinito	Atribuição de bits de ponto fixo - Dados3 (step)	
Transformação de coseno discreto inversa	Atribuição de bits de ponto fixo - Dados6 (pent)	
Transormação invesar Fast Fourier (Auto/Indust. Vers)	FFT de ponto fixo - Dados1	
Operação com matrizes	FFT de ponto fixo - Dados2	
Pointer Chasing	FFT de ponto fixo - Dados3	
Modulação de largura de pulso	Decodificador Viterbi GSM - Dados1 (get)	
Cálculo de velocidade na estrada	Decodificador Viterbi GSM - Dados2 (toggle)	
Pesquisa em tabela e interpolação	Decodificador Viterbi GSM - Dados3 (ones)	
Conversão de Tooth para Spark	Decodificador Viterbi GSM - Dados4 (zeros)	
Consumidor	**Rede**	**Escritório**
Compressão de JPGE	OSPF/Dijkstra	Dithering
Descompressão JPEG	Lookup de rotas/Patricia	Rotação de imagens
Filtro de escala de cinza	Fluxo de pacotes - 512 kbytes	Processamento de texto
Conversão RGB para CYMK	Fluxo de pacotes - 1 Mbyte	
Conversão RGB para YIQ	Fluxo de pacotes - 2 Mbytes	

Fonte: Autoria própria

Além dos arquivos com as especificações dos problemas para cada aplicação, a suíte também conta com seus algoritmos geradores. Desta forma, diferente do UUniFast, apesar do E3S fornecer os grafos de tarefas já prontos para o uso, ele possibilita a

customização dos arquivos geradores das tarefas em caso de necessidade do pesquisador, por disponibilizar os arquivos geradores dos grafos de tarefas para cada aplicação (DICK, 2008).

Cada *benchmark* apresenta vantagens e desvantagens na sua utilização. A seguir são listadas as vantagens de cada um. Primeiramente, o UUniFast por exemplo, apresenta como vantagem a possibilidade de definir como o *deadline* da tarefa e do conjunto como um todo deverá ser atendido, isso possibilita avaliar tarefas consideradas críticas, que devem ter seu deadline respeitado. Outra grande vantagem do UUniFast é o fato do usuário determinar apenas dois parâmetros fundamentais, o número de tarefas e a utilização da tarefa, não precisar se preocupar com outros parâmetros facilita o trabalho do pesquisador que pode focar no desempenho do seu escalonador. Vantagem essa que se estende ao E3S, pois ele já possui conjuntos inteiros de tarefas e processadores específicos para determinadas áreas prontos para serem utilizados. Além disso, outras vantagens do E3S, que também são vantagens do TGFF, é o relacionamento existente entre tarefas e processadores, que auxilia na pesquisa com sistemas embarcados e heterogêneos. Outro fator que engloba o TGFF e o E3S é a interdependência entre tarefas, pois na vida real, além das tarefas disputarem por recursos, elas também possuem prioridades e o escalonador precisa resolver essas questões de modo a respeitar todas as variáveis envolvidas. Finalizando, uma vantagem do TGFF é que ele possibilita ao pesquisador mais exigente gerar as tarefas de acordo com os seus requisitos de tipos de tarefas, tipos de processadores, tempos de execução e assim por diante. O TGFF fornece muitos parâmetros que podem ajudar na geração de grafos mais elaborados. Contudo, os benchmarks também apresentam desvantagens.

As desvantagens de cada um podem ser acompanhadas a seguir. A maior desvantagem do UUniFast, no contexto desse trabalho, é que ele não é viável em sistemas multiprocessados. Já a maior desvantagem do E3S é o número limitado de tipos de aplicações para as quais foram gerados os conjuntos de tarefas. Por outro lado, a maior desvantagem do TGFF, que também foi listada como vantagem, mas que pode retardar o processo de avaliação de um escalonador, é o fato dele exigir muitos parâmetros para geração de tarefas e se os parâmetros não forem específicos, pode

acarretar em resultados errôneos. Assim como já foi citado no parágrafo de vantagens, cada *benchmark* pode ser útil em alguns casos.

Cada uma das ferramentas tem aplicabilidades distintas e específicas. Utiliza-se, por exemplo, o UUniFast quando se quer uma distribuição mais uniforme do conjunto de tarefas, objetivando uma utilização maior dos recursos de processamento. Porém, as tarefas geradas pelo UUniFast não têm relação de dependência entre si. Para avaliar o desempenho de sistemas de tempo real com tarefas que possuem dependências, uma boa opção é utilizar o TGFF. Esta ferramenta gera grafo de tarefas que contém relações entre si e *deadline* compartilhado entre o nodo inicial do caminho do grafo até o nodo terminal. No entanto, ainda é possível evitar a configuração e parametrização destas ferramentas para geração de conjuntos de tarefas ao optar por utilizar a suíte E3S. Ela contém conjuntos de tarefas já gerados para os devidos fins. Ou seja, cada conjunto de tarefas tem uma função e o conjunto de tarefas foi gerado a partir das características de processadores do mercado para um determinado objetivo. Por exemplo, existem sistemas embarcados que são utilizados em redes de computadores. Esses sistemas, normalmente, executam algoritmos, como o *Djikstra* e outros, para encontrar o melhor caminho para entregar pacotes de dados. Portanto, o conjunto de tarefas do E3S para a finalidade de redes de computadores visa simular essas tarefas de acordo com parâmetros (tempo de operação, custo da operação, etc.) de processadores de mercado para este objetivo.

5 CONCLUSÃO

Finalmente, através da análise feita é possível verificar que existem ferramentas benchmark que avaliam diversos aspectos dos escalonadores de tarefas de tempo real em sistemas embarcados. A eficiência do escalonamento para um processador pode ser verificada pelo UUniFast, por outro lado a eficiência para um sistema multiprocessado pode ser medida pelo TGFF ou pelo E3S. Se o deadline é um fator importante para o pesquisador e se ele quiser medir a eficiência do escalonamento de cada processador, individualmente, de um sistema multiprocessado, então ele pode utilizar o UUniFast. Por outro lado, se o foco do pesquisador for verificar a eficiência do escalonamento em um sistema embarcado multiprocessado utilizando tarefas de uma das seguintes áreas: industrial/automobilística, consumidor, redes, escritório e telecomunicações, então ele deve utilizar o E3S. Além disso, o pesquisador pode alterar o E3S para atender tarefas de outras áreas.

Dentre os possíveis trabalhos futuros, destaca-se a execução do escalonamento das tarefas geradas em dispositivos embarcados multiprocessados, para verificar na prática, a eficiência do conjunto de tarefas. Além disso, pode-se também ampliar os benchmarks comparados e assim prover uma visão mais ampla das ferramentas para essa finalidade. Outra sugestão é comparar geradores de tarefas que visam avaliar o comportamento do escalonador frente ao consumo de energia da tarefa a ser escalonada.

REFERÊNCIAS

ALAHMAD, Bader N.; GOPALAKRISHNAN, Sathish. Energy efficient task partitioning and real-time scheduling on heterogeneous multiprocessor platforms with QoS requirements. **Sustainable Computing: Informatics and Systems**, v. 1, n. 4, p. 314-328, 2011.

AMD. What is Heterogeneous System Architecture (HSA)?. **Developer Central**. Disponível em < http://developer.amd.com/resources/heterogeneous-computing/what-is-heterogeneous-system-architecture-hsa/>. Acesso em 20 ago. 2016.

BINI, Enrico; BUTTAZZO, Giorgio C. Biasing effects in schedulability measures. In: **Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on**. IEEE, 2004. p. 196-203.

CHEN, Hua; CHENG, Albert Mo Kim; KUO, Ying-Wei. Assigning real-time tasks to heterogeneous processors by applying ant colony optimization. **Journal of Parallel and Distributed Computing**, v. 71, n. 1, p. 132-142, 2011.

CHEN, Jian-Jia; THIELE, Lothar. Energy-efficient task partition for periodic real-time tasks on platforms with dual processing elements. In: **Parallel and Distributed Systems, 2008. ICPADS'08. 14th IEEE International Conference on**. IEEE, 2008. p. 161-168.

CHEN, Jian-Jia; THIELE, Lothar. Platform synthesis and partitioning of real-time tasks for energy efficiency. **Journal of Systems Architecture**, v. 57, n. 6, p. 573-583, 2011.

CHIAPPETTA, Marco. How to *Benchmark* Your PC for Free. **PCWorld**.. 2 jul. 2012. Disponível em <http://www.pcworld.com/article/258473/how_to_*benchmark*_your_pc_for_free.html>. Acesso em 13 set. 2015

DICK, Robert. Embedded system synthesis *benchmark*s suites (E3S). 2008.

DICK, Robert P.; RHODES, David L.; WOLF, Wayne. TGFF: task graphs for free. In: **Proceedings of the 6th international workshop on Hardware/software codesign**. IEEE Computer Society, 1998. p. 97-101.

DI NATALE, Marco; DONG, C.; ZENG, H. Reality check: the need for benchmarking in RTS and CPS. In: proceedings of the 4th Real-Time Scheduling Open Problems **Seminar (RTSOPS'13)**. 2013. p. 18-19.

DORIGO, Marco; GAMBARDELLA, Luca Maria. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on evolutionary computation, v. 1, n. 1, p. 53-66, 1997.

EMBEDDED Computer Systems. **Electrical & Computer Engineering**. Disponível em: < http://www.ece.ncsu.edu/research/cas/ecs>. Acesso em: 25 nov. 2016.

EMBERSON, Paul; STAFFORD, Roger; DAVIS, Robert I. Techniques for the synthesis of multiprocessor tasksets. In: proceedings 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010). 2010. p. 6-11.

FARINES, Jean-Marie; FRAGA, Joni S.; OLIVEIRA, Rômulo S. **Sistemas de Tempo Real**. Universidade Federal de Santa Catarina, Departamento de Automação e Sistemas. Jul. 2000. Disponível em: < http://www.romulosilvadeoliveira.eng.br/livro-tr.pdf>. Acesso em: 12 jun. 2015.

FEOFILOFF, Paulo. O que é um grafo? **Algoritmos em Grafos**. 2 jun. 2015. Disponível em: <https://www.ime.usp.br/~pf/algoritmos_em_grafos/aulas/grafos.html>. Acesso em: 25 nov. 2016.

FEOFILOFF, Paulo. Grafos bipartidos versus circuitos ímpares. **Árvores e grafos bipartidos**. 23 set. 2016. Disponível em: < https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/bipartite.html >. Acesso em: 25 nov. 2016.

JELENKOVIĆ, Leonardo; GROŠ, Stjepan; JAKOBOVIĆ, Domagoj. **Testing task schedulers on Linux system**. 2009. Disponível em: < https://bib.irb.hr/datoteka/463774.cisse2009.pdf >. Acesso em: 29 nov. 2016.

KONG, Joonho; CHUNG, Sung W.; SKADRON, Kevin. Recent Thermal Management Techniques for Microprocessors. **ACM Computing Surveys**, New York (NY), v. 44, n. 3, article 13, 42 pages, jun. 2012. DOI = 10.1145/2187671.2187675

LAPLANTE, Phillip; Real-Time Operating Systems. **Institute of Electrical and Electronics Engineers**. Wiley-IEEE Press. p. 73 – 160. 2006. Disponível em: <http://ieeexplore.ieee.org.mutex.gmu.edu/xpl/ebooks/bookPdfWithBanner.jsp?fileName=5237075.pdf&bkn=5237056&pdfType=chapter>. Acesso em: 30 mai. 2015.

LOUREIRO, Antonio A. F. Terminologia. **Grafos**. 8 mar. 2015. Disponível em: <http://homepages.dcc.ufmg.br/~loureiro/md/md_9Grafos.pdf>. Acesso em: 25 nov. 2016.

MARIANI, Antonio C. Conceito Básicos da Teoria de Grafos. **Teoria dos Grafos**. Disponível em: <http://www.inf.ufsc.br/grafos/definicoes/definicao.html>. Acesso em: 25 nov. 2016.

MIMS, Christopher. Why CPUs Aren't Getting Any Faster. **MIT Technology Review**. 12 out. 2010. Disponível em: <https://www.technologyreview.com/s/421186/why-cpus-arent-getting-any-faster/>. Acesso em: 25 nov. 2016.

ROBBEN, Matthew. Interpreting CPU Utilization for Performance Analysis. Windows Server Performance Team Blog, 6 ago. 2009. Disponível em: <https://blogs.technet.microsoft.com/winserverperformance/2009/08/06/interpreting-cpu-utilization-for-performance-analysis/>. Acesso em: 24 nov. 2016

SILBERSCHATZ, Abraham et al. **Operating system concepts**. Reading: Addison-Wesley, 1998.

STEINBERG, Geoffrey; SANGHERA, Kamaljeet. Introduction to Computer Information Systems. Kendall/Hunt Publishing Co., 2007.

STEVENSON, Robert; Computers in the Military. **The University of North Carolina at Chapel Hill**. 8 Set. 2014. Disponível em <http://www.unc.edu/~rtsteven/comp101/project1/>. Acesso em: 23 nov. 2016.

UCHIYAMA, Kunio; ARAKAWA, Fumio; KASAHARA, Hironori; NOJIRI, Tohru; NODA, Hideyuki; TAWARA, Yasuhiro; IDEHARA, Akio; IWATA, Kenichi; SHIKANO, Hiroaki. Heterogeneous Multicore Architecture. **Heterogeneous Multicore Processor Technologies for Embedded Systems**. Springer (New York), p. 11-18, mar 2012. DOI = 10.1007/978-1-4614-0284-8_2. Disponível em: < http://link.springer.com.mutex.gmu.edu/chapter/10.1007/978-1-4614-0284-8_2 >. Acesso em: 4 jun. 2015.

VALENTIN, Eduardo B.; **smartenum**: Algoritmo Branch-And-Bound Para Determinação de Frequências Ótimas em Sistemas com Escala Dinâmica de Tensão e Frequência. 2010. Dissertação (Mestrado) - Programa de Pós-Graduação em Informática, Universidade Federal do Amazonas. Manaus, 2010.