Jonas Rossi Dourado

Projeto e aplicação de um cluster com processadores digitais de sinais

São Carlos - SP, Brasil

Junho de 2012

Jonas Rossi Dourado

Projeto e aplicação de um cluster com processadores digitais de sinais

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos, da Universidade de São Paulo

Orientador:
Carlos Dias Maciel

DEPARTAMENTO DE ENGENHARIA ELÉTRICA
ESCOLA DE ENGENHARIA DE SÃO CARLOS
UNIVERSIDADE DE SÃO PAULO

São Carlos - SP, Brasil

Junho de 2012

AUTORIZO A REPRODUÇÃO E DIVULGAÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

Dourado, Jonas Rossi
D739p Projeto e aplicação de um cluster com processadores digitais de sinais.. / Jonas Rossi Dourado ; orientador Carlos Dias Maciel –- São Carlos, 2012.

Monografia (Graduação em Engenharia de Computação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2012.

1. *Cluster*. 2. DSP. 3. MPI. 4. *Linux*. I. Titulo.

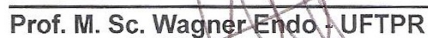
FOLHA DE APROVAÇÃO

Nome: Jonas Rossi Dourado

Título: "Projeto de um Cluster com Processadores Digitais de Sinais"

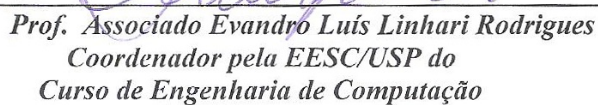
Trabalho de Conclusão de Curso defendido e aprovado em 25/06/2012,

com NOTA dez (10 , 0), pela comissão julgadora:



Prof. M. Sc. Wagner Endo - UFTPR

M. Sc. Jen John Lee - SEL/EESC/USP



Prof. Associado Evandro Luís Linhari Rodrigues
Coordenador pela EESC/USP do
Curso de Engenharia de Computação

Resumo

Atualmente, praticamente todas as áreas da ciência são beneficiadas a partir de um aumento no poder de processamento computacional. Quando não é possível obter um aumento de poder de processamento com a obtenção de processadores mais rápidos, frequentemente se recorre a utilização de vários processadores em conjunto, nos chamados *clusters*.

Esse trabalho apresenta como criar um *cluster* de processadores digitais de sinais (DSPs) utilizando a plataforma BeagleBoard. Os DSPs tem como grande benefício o elevadíssimo poder de processamento, principalmente quando se trata de cálculos brutos.

Abstract

Nowadays almost every area of science is favored with increased computer processing power. When it isn't possible to gain performance due to a better processor, often processors are used together in structures called clusters.

This document shows how to create a cluster using digital signal processors (DSPs) embedded on a BeagleBoard platform. The DSPs have as great benefit an elevated processing power, with highlight for raw computing power.

Dedicatória

Dedico este trabalho aos meus irmãos e pais que sempre estiveram ao meu lado. Amo todos: Amália, Breno, Sarah, Ângela e Marcos.

Dedico também aos meus Avós que muito me ensinaram sobre a vida.

E a todos que eu gostaria de passar mais tempo junto.

Agradecimentos

Agradeço antes de mais nada a Deus.

Agradeço aos meus amigos de república João Damasceno e Pedro Junão Bignatto por sempre estarem dispostos para uma boa conversa.

Aos meus amigos de turma que batalharam junto comigo durante o curso, atravessando longas noites de estudo, mas mesmo assim, nunca perdendo o bom humor. Entre eles, André Andrew Cunha, Bruno Brubru Begotti, Thiago Protege Roberto e Rafael Montanha Borges.

Aos meus amigos do Warthog Robotics, que deixaram minha graduação mais interessante. Sempre lembrarei das ótimas histórias e das maratonas pré competição alternadas entre chaves de fenda, linhas de código e café. Valeu Adriano Komesu, Bernardo Rodrigues, Eduardo Fraccaroli, Filipe Oliveira, Henrique Aguirre, Ivan Filgueiras, Gabriel Bombini, Leonardo Gomes, Lucas Topp, Lukas Reis, Pedro Carlson, Pedro Neves, Rafael Lang, Wesley Massuda e a todos os que não lembrei.

Ao meu orientador, inclusive de iniciação científica, Carlos Dias Maciel por sempre estar disposto a ajudar. Assim como o Maciel, o Saul Andrade foi essencial nos primeiros passos de uma possível vida acadêmica.

Por último e não menos importante, agradeço a todos os meus amigos de José Bonifácio pela amizade, fazendo de lá, a melhor cidade de todas! Não tentarei listarei todos porque tenho certeza que um ou outro ficaria de fora injustamente.

Conteúdo

Lista de Figuras

1 Introdução **11**

1.1 Contextualização 11

1.2 Objetivo deste trabalho 12

1.3 Estrutura da monografia 13

2 Fundamentação Teórica **14**

2.1 BeagleBoard 14

2.2 Arquitetura ARM 14

2.3 Arquitetura DSP 18

2.4 Sistemas Operacionais 18

2.5 Linux 21

2.6 Cross-Compilation 22

2.7 *Cluster* 23

2.8 Padrão MPI 25

2.9 Open-Embedded 26

2.10 Demais conceitos envolvidos 26

2.10.1 RS-232 26

2.10.2 Ethernet 27

2.10.3 USB 27

2.10.4 SSH 27

2.10.5 Speedup . . . 28
2.10.6 Emulador . . . 28

3 Materiais e Métodos **29**

3.1 Montagem do *hardware* necessário . . . 29
3.1.1 Alimentação . . . 29
3.1.2 Comunicação primária . . . 30
3.1.3 Rede . . . 31
3.1.4 *Hardware* opcional . . . 32
3.2 Criação de imagem para a BeagleBoard . . . 32
3.2.1 Pesquisa . . . 32
3.2.2 Ångström . . . 32
3.3 *Software* para o cluster . . . 33
3.4 Testes de funcionamento . . . 35
3.5 Criação da página estilo *Wiki* . . . 35

4 Resultados e Discussão **36**

5 Conclusões e trabalhos futuros **38**

Bibliografia **40**

Anexo A – Cálculo do π com o MPI **42**

Anexo B – Comandos para instalar o Ångström **44**

Anexo C – *Recipes* desenvolvidos durante o projeto **45**

Anexo D – *Shell script* usado para compilar o binário final **47**

Anexo E – *Shell script* usado para realizar os testes **49**

Lista de Figuras

2.1 BeagleBoard. . . . 15

2.2 Diagrama de blocos de um core Cortex-A8. . . . 16

2.3 Funcionamento de uma unidade de execução SIMD. . . . 17

2.4 Diagrama de blocos do DSP presente no OMAP3530. . . . 19

2.5 Interação entre usuário, programas, sistema operacional e *hardware*. . . . 20

2.6 *Kernel* monolítico. . . . 21

2.7 Micro *kernel*. . . . 21

2.8 O computador IBM 650 utilizava cartões perfurados 23

2.9 Desenvolvimento típico para sistemas embarcados. . . . 24

2.10 Ilustração de um *cluster* Beowulf. . . . 25

3.1 Esquemático de um cabo serial *null modem*. . . . 30

3.2 Ambiente utilizado para desenvolver o trabalho. . . . 31

3.3 Funcionamento da ferramenta C6RunLib. . . . 34

3.4 Geração de código com processamento feito no DSP. . . . 35

4.1 Resultados obtidos ao executar os testes. . . . 37

1 Introdução

1.1 Contextualização

Com o avanço da computação a ciência pôde encontrar uma grande ferramenta para auxiliar pesquisas de diversas maneiras. Um dos usos para a computação em praticamente todas as áreas da ciência é o da simulação.

A simulação permite criar virtualmente cenários governados por uma lógica similar o suficiente para imitar o funcionamento real [Smith 1998]. É possível simular coisas tangíveis como clima, comportamento de multidões ou iteração entre corpos celestes, além de coisas mais abstratas como iterações entre partículas quânticas ou funcionamento de neurônios.

Durante a definição da lógica da simulação, ao se tomar decisões, há a necessidade de se optar por um maior detalhamento ou por um menor tempo de processamento. Ou seja, há um *trade-off* constante entre precisão e velocidade de resposta. Comumente a precisão é limitada artificialmente pelos cientistas devido exclusivamente ao fato de que aumentar a precisão pode deixar o processamento demorado de maneira que o resultado não seja gerado em tempo razoável para o fim desejado.

Um exemplo desse *trade-off* é visível em uma simulação climática em que a iteração física entre todas as moléculas do ar, o sol e a camada terrestre é realizada. Uma simulação com essa configuração é impraticável em um tempo razoável com o poder de processamento disponível atualmente. Para driblar a limitação computacional, os cientistas geralmente abstraem tal precisão simulando por exemplo apenas as iterações de frentes de ar e correntes de convenções marítimas.

Como artifício para conseguir melhor precisão na simulação e não aumentar consideravelmente o tempo de processamento, é lógico pensar em se obter processadores mais rápidos. Entretanto, conseguir processadores mais rápidos possui limites, um deles é devido a natureza do transistor (componente base para dos processadores), que ao aumentar a sua velocidade, aumenta linearmente o seu consumo de energia [Sedra e Smith 2009]. Para diminuir o consumo

é preciso diminuir o tamanho dos transistores, o que nem sempre é possível com a tecnologia vigente. O outro limitante é que muitas vezes o custo de um processador melhor não compensa o ganho de processamento, tornando a opção economicamente inviável.

Para circundar essa limitação dos processadores individuais, há a possibilidade de colocar mais de um processador para trabalhar em conjunto, em uma estrutura que é chamada de *cluster* [Baker e Sterling 2000]. A utilização de *clusters* é amplamente difundida na área científica, com vários centros de pesquisa ao redor do mundo possuindo um. Essa é apenas uma das aplicações de *clusters*, sendo que qualquer processamento pesado que possa ser paralelizável pode se beneficiar dessa estrutura, entre eles:

- Processamento Digital de Sinal;
- Mineração de dados;
- Sistemas de tempo real;
- Processamento pesado em geral.

Em específico esse trabalho, devido ao uso de processadores de baixo consumo energético, torna possível criar *clusters* embarcados, que podem ter as mais diversas aplicações práticas, como equipamentos médicos portáteis de diagnósticos com resultados instantâneos ou processamento de dados de UAVs (*Unmanned Aerial Vehicles* - Veículos Aéreos Não Tripulados) em tempo real.

Outro ponto que pode ser explorado a partir desse trabalho é *grid computing* [Foster 2002], um termo relativamente novo, que significa usar aparelhos ociosos para executar processamento distribuído (Geralmente o termo *grid computing* se refere a sistemas menos acoplados e dispersos geograficamente). Pode ser encontradas versões parecidas do processador utilizado no projeto em celulares disponíveis comercialmente que rodam Android [MOTOROLA]. Uma adaptação do sistema utilizado nos celulares [1] para torná-los aparelhos aptos a participar de um *Grid* não é muito complexa.

1.2 Objetivo deste trabalho

O trabalho é uma continuação dos projetos desenvolvidos [Araújo, Dourado e Maciel 2010] [Dourado e Maciel 2011] no Laboratório de Processamento de Sinais (LPS) relacionados a cri-

[1] Os celulares com Android utilizam Linux, o mesmo sistema operacional no qual o trabalho é desenvolvido.

ação de um *cluster* com Processadores Digitais de Sinais (DSPs). O principal objetivo é documentar de forma sistemática o processo de transformar BeagleBoards [Coley 2012] em um *cluster*. Como um adicional aos trabalhos anteriores, é a criação uma distribuição customizada de Linux através do uso da ferramenta Open Embedded.

Para realizar o objetivo, o planejamento inicial consiste em:

- Montagem do *hardware* necessário;
- Criar uma imagem Linux para as BeagleBoards;
- Compilar e configurar *software* para *cluster*;
- Testes de funcionamento;
- Criação de uma página no estilo Wiki.

1.3 Estrutura da monografia

A estrutura da monografia foi dividida entre os capítulos Introdução, Fundamentação Teórica, Materiais e Métodos, Resultados e Discussão, Conclusões e Trabalhos futuros e Bibliografia. Ao final da monografia foi acrescentados anexos pertinentes ao trabalho.

2 Fundamentação Teórica

Neste capítulo é apresentado a fundamentação teórica dos conceitos envolvidos ao realizar o trabalho.

2.1 BeagleBoard

A BeagleBoard [Coley 2012] apresentada na Figura 2.1 é uma placa para desenvolvimento de sistemas embarcados com licenciamento *open-source*. O baixíssimo consumo da plataforma, o tamanho reduzido de cerca de 7,2 cm X 7,2 cm e o poder de processamento fornecido pelo microcontrolador OMAP3530, são características de um sistema com grande potencial para se montar um *cluster* de baixo consumo e grande desempenho.

O processador OMAP3530 [Texas Instruments 2009] da Texas Instruments é um processador *multi core* híbrido, com um processador ARM Cortex-A8 de 720 MHz, um processador digital de sinal de ponto fixo TMS320C64x+ de 520 MHz e um processador de vídeo POWERVR SGX. A BeagleBoard possui 256 MB (ou 128 MB na revisão B) de memória RAM e 256 MB de memória ROM.

Como entrada e saída, a placa possui suporte para saída de video (HDMI ou S-Vídeo), entrada/saída de áudio, *slot* para cartão SD, serial RS-232, *headers* de JTAG e USB on-the-go.

O fornecimento de energia para a BeagleBoard é através de um conector *jack* alimentado com a tensão de 5V e o consumo é de até 2W.

2.2 Arquitetura ARM

A arquitetura ARM teve sua origem quando a empresa britânica Acorn Computers Ltd no início da década de 80 precisava de um processador capaz de substituir o microprocessador 6502, presente nos seus populares (Na Inglaterra) computadores *BBC micro*. Ao realizar teste

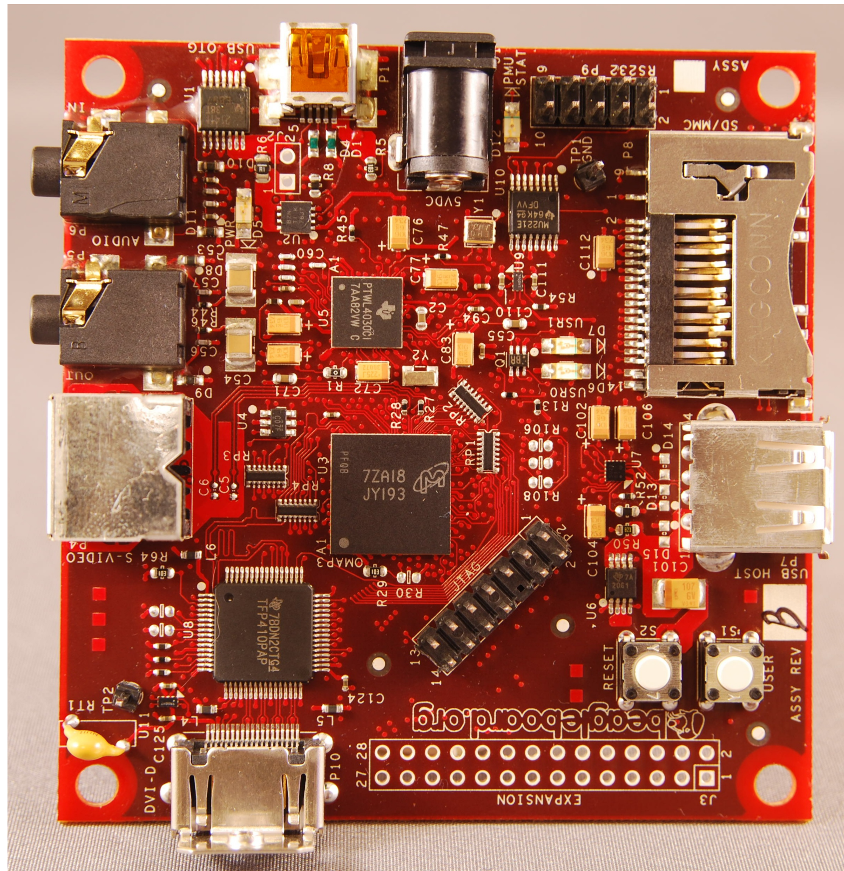


Figura 2.1: BeagleBoard.

com os micropocessadores disponíveis no mercado, não encontrou algum que satisfizesse as especificações procuradas. [Carol Atack e Alex van Someren 1993]

Embora a Acorn não estava no mercado de ASICs (*Application-specific integrated circuit*), a grande quantidade de caixa resultante do sucesso do *BBC micro* permitiu a extravagância de investir a fundo perdido na criação de microprocessador próprio. Juntando um time de engenheiros talentosos entretanto inexperientes, conseguiu definir e criar o ARM1 (À época, acrónimo para ***A**corn **RISC** **M**achine*), o que foi considerado um grande feito diante da inexperiencia da equipe.

Com um relativo sucesso até o ARM3, a arquitetura chamou a atenção da Apple e da VLSI Technology, que ajudaram a fundar a *spin-off* ***A**dvanced **RISC** **M**achines Ltd*, rebatizando o significado do acrónimo ARM. Atualmente o modelo de negócios da ARM Holdings (Nome atual) é a venda de licenças IP (*Intellectual Property*) das diferentes versões da aquitetura e então as empresas que compram as licenças, criam circuitos integrados (Exemplo: SoCs, *System-on-Chips*) com o núcleo de processamento ARM.

Atualmente a arquitetura ARM encontra-se fortemente difundida, principalmente no seg-

mento de sistemas embarcados, onde já em 2010, já era responsável por 95% do *market share* de *smartphones* e hoje em dia, corresponde mias de 75% de todos os microprocessadores embarcados de 32 *bits*. [Timothy Prickett Morgan 2011]

Um dos motivos que torna a arquitetura ARM eficiênte em termos energéticos é a opção por um conjunto reduzido de instruções, um modelo chamado de RISC (*Reduced Instruction Set Computer*) em detrerimento ao modelo CISC (*Complex Instruction Set Computer*). A adoção de um pequeno número de instruções permitiu criar um chip com poucos transistores (o que influencia diretamente no consumo), facilitando assim o desenvolvimento e teste.

O processador OMAP3530 utilizado na BeagleBoard possui como um dos núcleos um processador ARM *Cortex-A8* que tem seu diagrama de classe representado na Figura 2.2. Além de implementar a arquitetura ARMv7, entre suas características, se destacam um *pipeline superescalar*, uma unidade de execução SIMD (*Single instruction, multiple data*), Thumb-2 e Jazelle RCT (*Runtime Compilation Target*). [ARM Limited 2011]

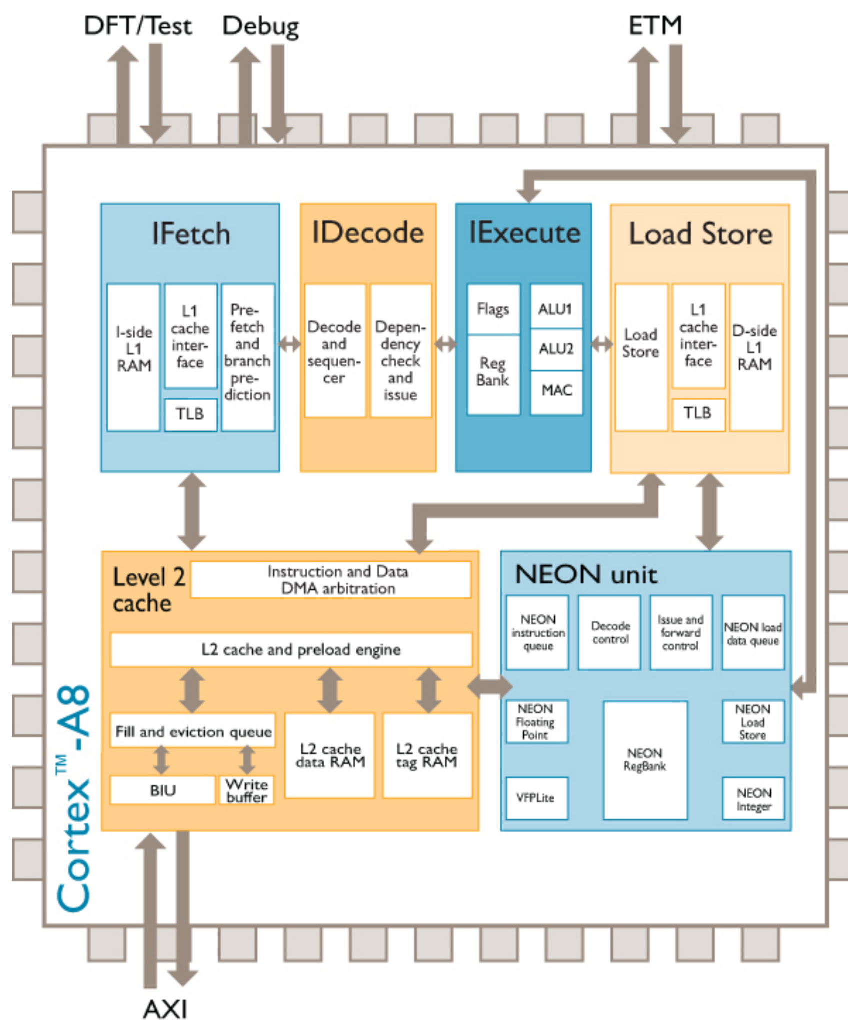


Figura 2.2: Diagrama de blocos de um core Cortex-A8.

O *pipeline superescalar* permitiu um aumento de duas vezes no número de instruções executadas por *clock* em relação ao ARM11. A unidade de execução SIMD, comercialmente chamada de NEON, estende a conjunto de instruções, permitindo a execução de uma instrução paralelamente em múltiplos pedaços de dados ao mesmo tempo, um exemplo de unidade SIMD é apresentada na Figura 2.3.

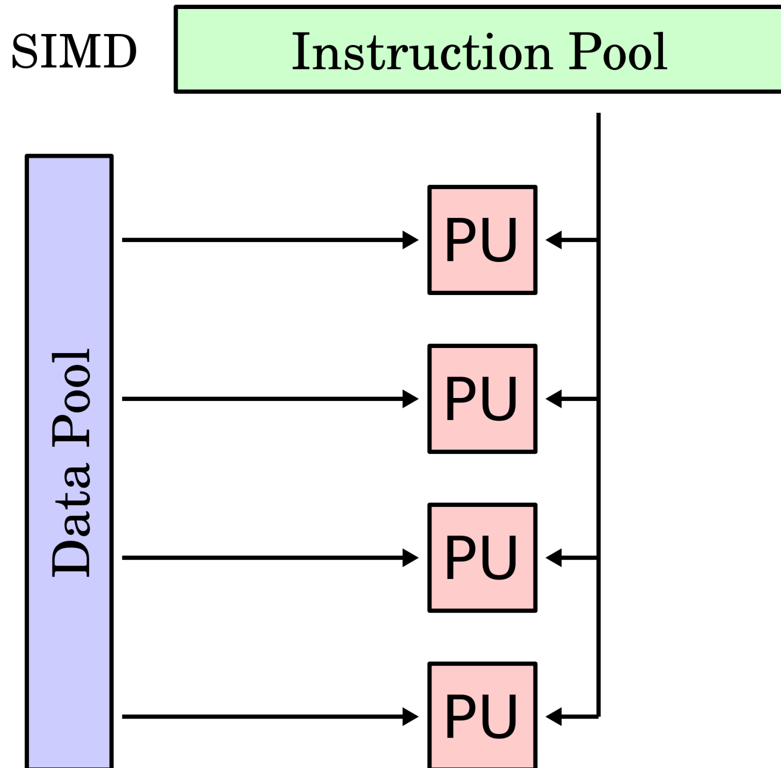


Figura 2.3: Funcionamento de uma unidade de execução SIMD.

A tecnoglogia Thumb-2 que teve estréi na arquitetura ARMv7 permite o aumento da densidade de códido através da utilização de instruções de tamanho variáveis, podendo ser de 16bits ou 32bits. Não há perda de desempenho ao utilizar instruções 16bits pois o processador transforma em hardware as instruções de 16bits em equivalentes de 32bits, o que evita também a adição de muitos transistores já que não há duplicação de funções em *hardware*. A tecnoglogia Jazelle RCT foi introduzida nos *cores* Cortex-A8 com o intuito de aumentar o desempenho de aplicações que fazem o uso de compilação JIT (*just in time*), como Java e Python.

2.3 Arquitetura DSP

A arquitetura DSP (*Digital signal processor*) surgiu da necessidade de realizar processamento de sinais analógicos que inevitavelmente acabam dependendo de grande poder de processamento. Antes da sua criação (final da década de 70), o tratamento de sinais era implementado utilizando processadores *bit-slice*, onde vários circuitos integrados eram interligados para funcionar de acordo com as necessidades do projeto. [IEEE Milestones]

Com o aumento da necessidade de fazer processamento digital de sinal, foi lógica a criação de processadores especializados, sendo o primeiro o TMS5100 produzido pela Texas Instruments em 1978 para um brinquedo que sintetizava fala e consequentemente precisava de processamento especializado de sinal.

Desde a criação do primeiro DSP até os dias de hoje, a poder de processamento teve um aumento significante, sendo que enquanto no início uma das instruções mais utilizadas em um DSP, a MAC (*Multiply-accumulate operation, Ex.* $a \leftarrow a + (b \times c)$) durava 390 ns para ser calculada, atualmente é possível realizar essa mesma operação em cerca de 3 ns.

Especificamente na BeagleBoard, é utilizado um core de DSP TMS320C64x+ (Figura 2.4 da Texas Instruments, que utiliza arquitetura Harvard para acesso a memória. A arquitetura Harvard se diferencia da von Neumann no fato de possuir memórias (assim como o mecanismo de acesso) distintas para instruções e dados enquanto que a von Neumann acessa as memórias de instrução e dados pelo mesmo barramento. A arquitetura Harvard tem como principal vantagem poder ser mais rápido já que não há concorrência de barramento entre as memórias de instrução e dados.

O *core* TMS320C64x+ possui paralelismo em nível de instrução através de instruções VLIW (*Very Long Instruction Word*) de 256 bits, em que cada instrução é na verdade um conjunto delas. [Texas Instruments 2010] No processador usado no trabalho, é possível fazer até oito operações com número de 32 bits. Os dados são carregados por dois *data paths*, o que permite trabalhar com endereços distintos a cada acesso memória.

2.4 Sistemas Operacionais

Com o aumento da complexidade dos sistemas computacionais, houve a necessidade de facilitar o desenvolvimento e gerenciamento de aplicações que executam nos processadores. Visando tal fim foi criado os sistemas operacionais, que são na verdade um conjunto de *software* com a função de gerenciar recursos e fornecer serviços, ou seja, atuam como uma camada

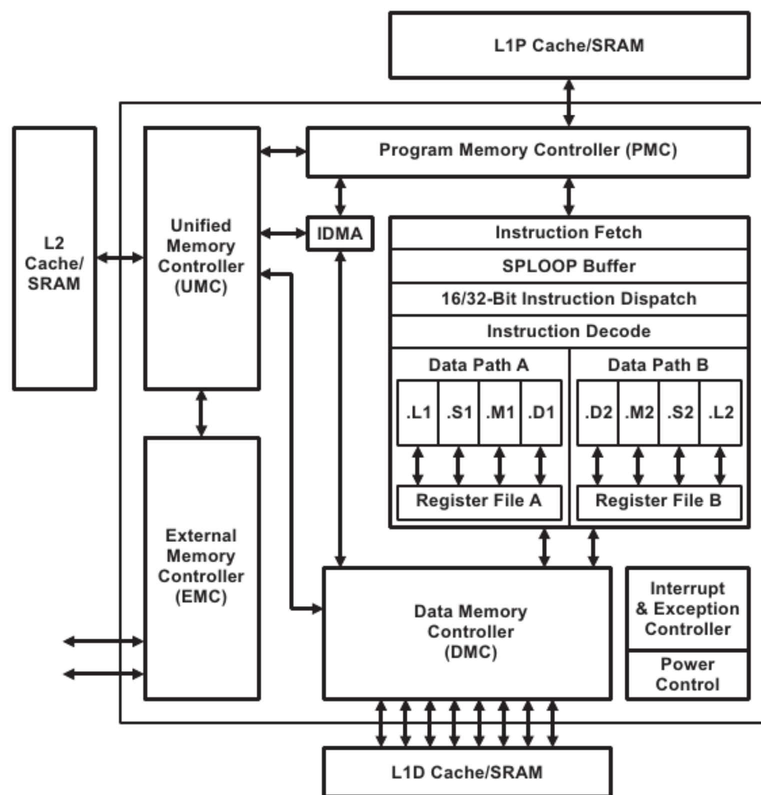


Figura 2.4: Diagrama de blocos do DSP presente no OMAP3530.[Texas Instruments 2010]

entre o *hardware* e o *software* executado pelo usuário. Um esquema simplificado da interação entre usuário, programas, sistema operacional e *hardware*, pode ser observado na Figura 2.5. [Stallings 2008]

Os sistemas operacionais modernos permitem o compartilhamento de recursos entre vários programas através de um mecanismo chamado de *Time-Sharing*. O *Time-Sharing* consiste em dividir o uso da CPU e de outros recursos com o uso de espaços de tempo (*Time slots*) em que cada programa possui a exclusivamente a CPU disponível. O gerenciamento de qual programa vai executar e por quanto tempo é decidido por um sub sistema chamado de escalonador de processos.

O compartilhamento de recursos implica em poder separar os recursos de forma que uma parte do recurso alocada para um programa, não interfira em outro. Entre os mecanismos utilizados, um dos principais é o gerenciador de memória. O gerenciador de memória é responsável por separar pedaços de memória para cada programa e ao mesmo tempo, não permitir que um programa acesse memória que não seja a própria.

Durante a execução dos processos há a necessidade de realizar operações de entrada e saída de programas. Para realizar entrada e saída antes de mais nada é preciso um meio de fazer as informações serem trocadas com os periféricos conectados ao *hardware*, o que é feito utilizando

Figura 2.5: Interação entre usuário, programas, sistema operacional e *hardware*.

device drivers (programas que permitem os dispositivos conectados ao sistema funcionarem corretamente). Os *device drivers* implementam particularidades específicas de cada *hardware*, o que permite usar uma padrão de acesso a todos os dispositivos de forma transparente para um desenvolvedor que utilize o sistema.

O nome dado ao núcleo do sistema, que engloba o gerenciamento de memória, processador e o uso de *device drivers*, é *kernel*. O *kernel* pode ser implementado a princípio como monolítico, *micro kernel* ou híbrido.

O *kernel* monolítico (Figura 2.6) tem esse nome pois todos os sub sistemas são executados no mesmo espaço de memória (todos os sub sistemas do *kernel* tem acesso a memória dentre eles). As principais vantagens do *kernel* monolítico comparado ao *micro kernel* é o desempenho e o tamanho reduzido (uma consequência do tamanho reduzido é um menor número de erros no código).

O *micro kernel* (Figura 2.7) busca implementar apenas o mínimo necessário para suportar a execução de programas que atuam como servidores de serviços. Geralmente é implementado o mínimo de gerenciamento de memória, comunicação entre processos e tratamento de interrupções. Os serviços fornecidos pelos servidores que rodam em *user space* (Os servidores não enxergam a memória um dos outros) podem ser *device drivers*, sistema de vídeo, sistemas arquivos entre outros. As vantagens podem ser resumidas em uma manutenção mais simples

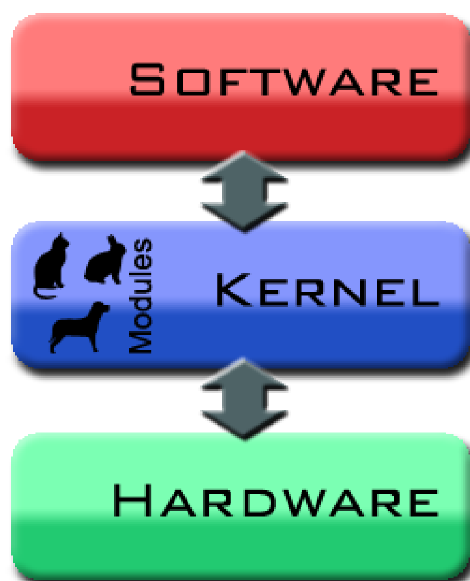


Figura 2.6: *Kernel* monolítico. [Ysangkok 2007]

e uma maior facilidade de desenvolvimento (não há a necessidade de reiniciar o sistema para testar uma mudança no código).

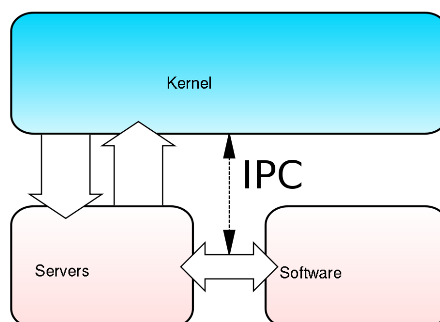


Figura 2.7: Micro *kernel*. [Vikketorr 2010]

O *kernel híbrido* é uma combinação em qualquer proporção entre *micro kernel* e *kernel* monolitico e atualmente é um dos mais utilizados. Exemplos de sistemas que utilizam *kernel* monolítico são os BSDs, AIX, HP-UX, Linux e MS-DOS. Para exemplos *micro kernel* podemos citar MINIX e o Mach (posteriormente adaptado e usado no MAC OS X). Por final, podemos citar como *kernel* híbrido o NT kernel (Utilizado no Windows 2000 em diante) e o MAC OS X.

2.5 Linux

No início da década de 90, o então estudante da *Helsinki University of Technology* Linus Torvalds, começou a fazer um sistema operacional por *hobby* com a intenção de aproveitar os

recursos de seu computador. Após desenvolver anonimamente, lançou o código na internet em 1991, dando início a um modelo de desenvolvimento aberto. Inicialmente o nome adotado foi Freax, entretanto o administrador de servidor FTP (File Transfer Protocol) que hospedava o código fonte, não gostando do nome, renomeou para Linux. Linus acabou consentindo com a troca de nome. [Torvalds e Diamond 2001]

O Linus utilizou componentes de terceiros (em especial os do projeto GNU) para desenvolver e executar o sistema Linux. Hoje em dia, o papel de cuidar dos programas que são inclusos junto com o *kernel* Linux, é dos mantedores das diversas distribuições. Uma distribuição é o conjunto do *kernel* mais os programas selecionados que tem como intuito atender uma determinada necessidade, como uso em servidores web ou cálculo científico.

Desde o começo, o Linux foi implementado como um *kernel* monolítico, tendo suporte a tarefas concorrentes (*multi threading*) e multi usuário. Com o tempo foi adicionado suporte a outras arquiteturas além da x86 inicialmente existente, sendo hoje, relativamente simples de portar para novas arquiteturas.

O Linux é utilizado por diversas organizações, como Google, Facebook e IBM. As aplicações são as mais variadas como celulares, servidores web, roteadores, televisões e super computadores.

2.6 Cross-Compilation

Os primeiros computadores eram programados de forma muito mecânica (como exemplo, um computador IBM 650 é apresentado na Figura 2.8), sendo preciso primeiro entender o funcionamento do *hardware* antes de começar a programar. Com o avanço do *hardware*, criou-se a possibilidade de fazer programas mais complexos e consecutivamente a necessidade de facilitar a maneira de se programar.

Visando automatizar o processo mecânico inicialmente foi desenvolvido os *assemblers* que são programas que interpretam cada mnemônico escrito e transcreve para a instrução correspondente em binário. Embora um avanço significativo, ainda havia a necessidade de saber o conjunto de instruções da arquitetura utilizado pelo *hardware*.

Como maneira de se abstrair o funcionamento do processador em relação a lógica do programa, foi criado as linguagens de programação e os compiladores. Os compiladores transformam expressões de uma linguagem de programação (linguagem de alto nível) em código de máquina (linguagem de baixo nível), executável no processador. De maneira sucinta, o compila-

Figura 2.8: O computador IBM 650 utilizava cartões perfurados

dor interpreta o arquivo com o código de uma linguagem de nível maior, realiza otimizações em um código intermediário e então gera um arquivo com código de nível mais baixo (geralmente um binário com código de máquina).

A compilação cruzada (Figura 2.9) ou mais comumente chamada de *cross-compilation*, surgiu da necessidade de se gerar código de máquina para máquinas com pouco poder de processamento, onde não é possível desenvolver nativamente. [GNU 2012] O *cross-compilation* é especialmente útil no desenvolvimento de sistemas embarcados, onde uma máquina com maior poder de processamento é utilizada para o desenvolvimento. O *cross-compiler* executado na maquina de desenvolvimento (chamado de *host*) lê o código fonte do programa e então gera um executável compatível com a arquitetura do processador de destino (*target*).

2.7 *Cluster*

Através da utilização de sistemas computacionais tradicionais (alguns núcleos de processamento por máquina), há basicamente duas possibilidades de se obter um ganho de desempenho, uma delas é uma maior otimização do código e a outra, um aumento do desempenho do processador em si. [Baker e Sterling 2000]

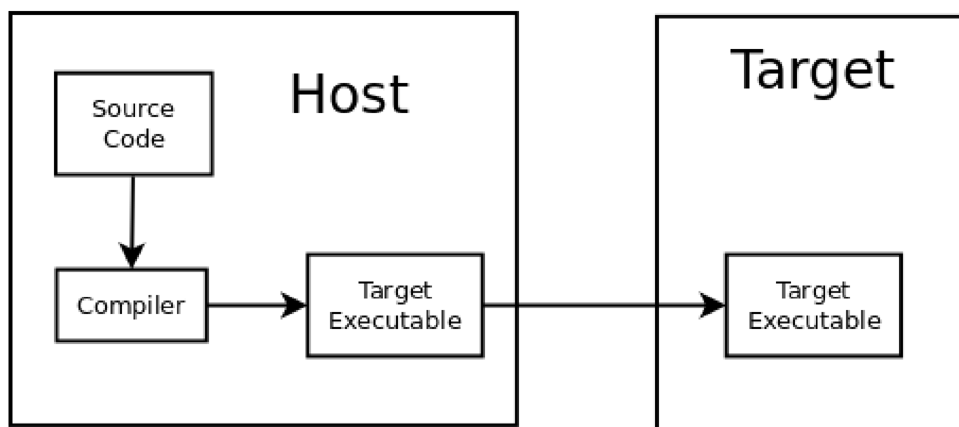


Figura 2.9: Desenvolvimento típico para sistemas embarcados.

O limitante tecnológico de se aumentar o desempenho dos processadores é devido ao consumo energético dos processadores (especificamente dos transistores) aumentar quadraticamente em relação a velocidade (*clock*) do processador [1]. Outra possibilidade de aumento de desempenho por processador é aumentar o número de núcleos ou melhorar o *pipeline* dos processadores, entretanto, existe o mesmo problema em termos de dissipação térmica. Por fim, há limitação de natureza econômica pois um aumento de poder de processamento pode ter custos desproporcionais ao aumento desempenho.

No que tange a otimização de código executado nos processadores, há sempre um limitante, pois nem sempre é possível melhorar o desempenho de uma aplicação devido as particularidades intrínsecas da arquitetura, a aplicação já estar otimizada no seu nível máximo ou devido ao custo da otimização ser proibitivo.

Como forma alternativa de aumentar o desempenho de sistemas computacionais, foi criado os *clusters* de computadores, que são conjuntos de computadores (chamados de nós) conectados entre si por alguma interface de comunicação. Uma grande vantagem de se optar por utilizar *clusters* é a flexibilidade na implementação e expansão (por exemplo, ao aumentar o número de nós). Os *clusters* podem ser montados desde máquinas projetadas especificamente para clusterização ou por *hardware* padrão de computadores pessoais, utilizando inclusive rede *Ethernet*. A um *cluster* utilizando componentes de computadores pessoais, software para processamento distribuído e com gerenciamento centralizado, pode ser chamado de um *cluster* Beowulf, ilustrado na Figura 2.10.

Um cluster Beowulf geralmente é implementado com a intenção de se obter um grande poder de processamento, com um bom custo-benefício. Geralmente utiliza computadores pes-

[1] O que pode ser contornado usando processos de fabricação menores, o que representa também um limitante tecnológico

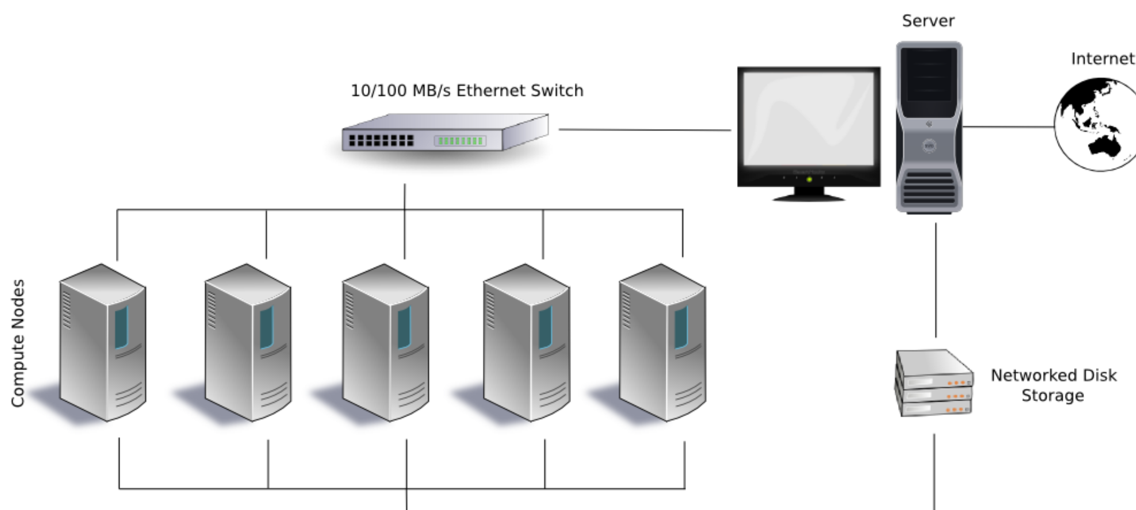


Figura 2.10: Ilustração de um *cluster* Beowulf.

soais baratos (Em relação a *hardware* para servidores), com rede *Ethernet* e como software, um sistema *Unix-like* aberto, com bibliotecas que possibilitam o compartilhamento de recursos computacionais, como alguma implementação do MPI. O gerenciamento de um *cluster* Beowulf é feito utilizando arquitetura de mestre-escravo, onde um dos nós é responsável por distribuir a tarefa entre os nós.

2.8 Padrão MPI

O padrão Message Passing Interface (MPI) é uma especificação de um sistema de troca de mensagens em sistemas distribuídos, desenvolvido pela indústria e academia. O desenvolvimento acorreu devido a necessidade de se padronizar as implementações de passagem de mensagens em sistemas distribuídos, pois inicialmente cada empresa ou grupo acadêmico definia a sua maneira de que jeito seria a passagem de mensagens de seu sistema. Atualmente o MPI se tornou um padrão *de-facto* para sistemas de memória distribuída (*clusters*). [Snir et al. 1998]

A especificação define um protocolo de comunicação independente de uma linguagem de programação, com funções de comunicação ponto a ponto e em grupo. Entre as funções há en-

tre outras, enviar/receber dados, barreiras de sincronização, combinação de resultados (*gathers* e *reduces*), número de processadores e processador atual. No trabalho foi utilizada a implmentação MPICH2 do Argonne National Laboratory.

No exemplo apresentado no Anexo A, há o cálculo da constante matemática π através da integração numérica da Função 2.1. Primeiramente o nó servidor distribui entre os nós escravos através de um *broadcast* o tamanho do passo da integração numérica, após isso todos os nós (inclusive o servidor) calcula uma parcela da integração numérica e finalmente o servidor recolhe os resultados aplicando uma função de soma entre eles (*reduce*) e apresenta o tempo de execução.

$$f(x) = \int_0^1 \frac{4}{1+x^2} \mathrm{d}x \tag{2.1}$$

2.9 Open-Embedded

O Open-Embedded é uma *framework* para criação de imagens customizadas de Linux embarcado, permitindo aos desenvolvedores criar um distribuição completa de Linux selecionando os programas ou conjuntos de programas desejados. Possui um estrutura modularizada através de *Layers* que são camadas que podem se sobre escrever e/ou ser usadas de maneira hierárquica.

Entre as principais vantagens é o suporte a diversas arquiteturas e a enorme quantidade de *recipes* de programa disponíveis. Para criar uma distribuição de Linux, é preciso criar um *Layer* com os arquivos de configuração. Um *Layer* é basicamente um diretório contendo *recipes* e um arquivo *./conf/layer.conf* com as configurações do *layer*.

O Open-Embedded utiliza a ferramenta BitBake para interpretar e executar as tarefas dos *recipes*. O BitBake foi derivado do Portage, o gerenciador de pacotes da distribuição de Linux Gentoo.

2.10 Demais conceitos envolvidos

Nessa seção é apresentado conceitos envolvidos no desenvolvimento que são secundários em termos de realização do trabalho.

2.10.1 RS-232

O padrão RS-232 mantido pela *Electronic Industries Alliance* (EIA) em conjunto com a *Telecommunications Industry Association* (TIA) define normas para comunicação serial. O padrão

RS-232 engloba características elétricas, temporização de sinais, significado de sinais, tamanho e padrão de pinagem dos conectores. [Electronic Industries Association. Engineering Dept 1969]

2.10.2 Ethernet

Ethernet é um conjunto de padrões de comunicação padronizado a partir de 1985 pelas normas 802.3x da *Institute of Electrical and Electronics Engineers* (IEEE), que define a camada física e a de enlace de redes cabeadas (podendo ser de cobre ou óticos). Os padrões mais comuns atualmente são os de 100 Mbps e de 1 Gbps. [Metcalfe e Boggs 1976]

2.10.3 USB

O barramento *Universal Serial Bus* (USB) [2] é um padrão de comunicação criado pela industria para padronizar a comunicação de periféricos. É o substituto direto de varias interfaces como porta paralela, porta serial e PS/2.

Existem 3 versões diferentes no mercado, sendo que elas são retro compatíveis. A versão 1.1 possui duas classes de velocidade, a *low speed* de 1,5 Mbps e a *full speed*, de 12 Mbps. A versão 2.0 além de adicionar outra classe de velocidade (*full speed*, de 480 Mbps), incluiu melhorias como conectores menores e a possibilidade de dispositivos poderem funcionar tanto como *host* ou como *device*, chamada de USB *On-The-Go*. [USB IMPLEMENTERS 2012]

Por fim, a versão 3.0 aumentou a velocidade máxima para 5 Gbps (*super speed*) e o máximo fornecimento de corrente para cada porta para 900 mA (Ante 500 mA do USB 2.0).

2.10.4 SSH

O *Secure Shell* (SSH) é um protocolo de rede que possibilita a troca de dados, acesso a linha de comando remota e execução de comando remoto. Com o SSH é possível conectar através de um interface de rede em um computador que esteja executando um servidor de SSH. [Ylonen e Lonvick 2006] Ao se conectar no servidor, o usuário enxerga o sistema ao qual se conectou como se fosse um computador local, podendo executar qualquer tipo de comando que tenha permissão. O MPICH, a implementação do protocolo MPI utilizado nesse trabalho usa o SSH para comunicação entre os nós do *cluster*.

[2] Na verdade não é um barramento propriamente dito, pois os *devices* tem acesso apenas a própria linha de dados

2.10.5 Speedup

O Speedup é um termo utilizado em computação paralela que significa o ganho de desempenho de um programa executado de forma serial em relação ao mesmo programa executado de forma paralela. A definição matemática para o para o Speedup é apresentado na Equação 2.2

$$S_p = \frac{T_1}{T_p} \tag{2.2}$$

Na Equação 2.2, S_p representa o Speedup em si, T_1 é o tempo de execução serial do programa e T_p é o tempo de execução para o mesmo programa rodado em p processos concorrentes.

2.10.6 Emulador

Emulador é um *software* que simula um ambiente permitindo que um *software* criado para uma determinada plataforma seja executado em um plataforma diferente. Entre as principais vantagens de se utilizar emulador é poder testar *software* criado para uma plataforma específica em outra alem permitir a utilização de códigos legados em *hardware* moderno. [Koninklijke Bibliotheek]

No projeto foi utilizado o emulador QEMU, capaz de emular diversas arquiteturas e sistemas completos.

3 *Materiais e Métodos*

Esse capítulo descreve todos os passos realizados na montagem de um *cluster* de DSP utilizando BeagleBoards.

3.1 Montagem do *hardware* necessário

O primeiro passo para se criar um *cluster* de BeagleBoards é providenciar o *hardware* que provém suporte para o funcionamento correto da placa.

3.1.1 Alimentação

Para ligar a alimentação da placa é necessário uma tensão de 5 V e pelo menos 500 mA de corrente disponível. Anteriormente a esse trabalho, a alimentação era obtida através de uma fonte de corrente existente no laboratório. Mas devido a falta de disponibilidade, foi necessário obter uma corrente regulada de 5 V de outra fonte. Sabendo que a especificação elétrica do barramento USB era compatível, o barramento USB foi escolhido como nova fonte de energia. Para adaptar o conector USB a uma saída compatível fisicamente com os conectores fêmea do tipo banana (idêntico ao conector anterior da fonte de corrente), um cabo A-B (Comum de impressoras e outros periféricos) foi cortado ao meio, mantendo a metade com o cabo com conector A e descartando a metade com o conector B. Na extremidade desencapada, foi adicionado conectores fêmea do tipo banana e fixados em um papelão rígido.

O cabo com conector macho do tipo banana em uma extremidade e conector *jack* de energia na outra extremidade foi herança dos projetos anteriores que utilizavam a BeagleBoard e explica a opção pela manutenção do conector do tipo banana como padrão de conexão para alimentação.

3.1.2 Comunicação primária

Embora a BeagleBoard possua saída de vídeo, a primeira interação com a placa e única possível para depurar o processo de *boot* é a serial RS-232. Novamente, devido a falta de disponibilidade de uma porta serial RS-232 (os computadores atuais não trazem mais uma conector desse tipo), foi necessário adquirir um adaptador RS-232-USB. Após ler a documentação, foi constatado a necessidade de um cabo serial *null modem* como mostrado no esquemático na Figura 3.1 para fazer a conexão entre a BeagleBoard e a porta RS232. Como maneira simples de realizar a conexão, foi utilizado *jumper wires* para conectar o adaptador diretamente ao *pin header* de serial da BeagleBoard.

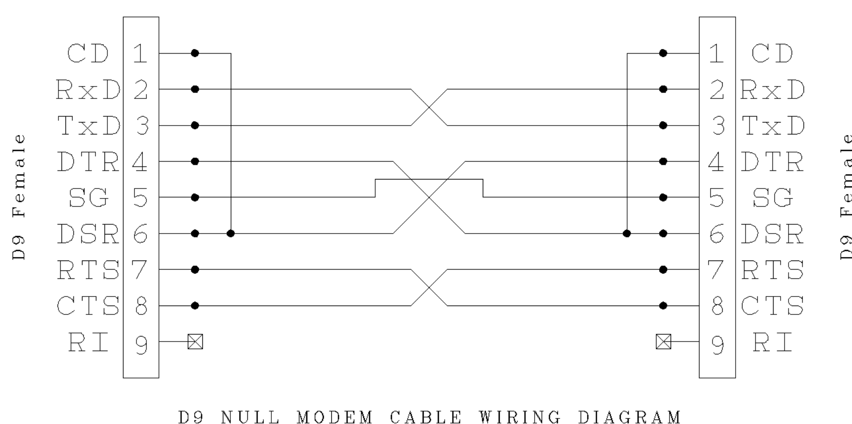


Figura 3.1: Esquemático de um cabo serial *null modem*.

Ao ligar a placa e observar a saída serial, observou-se muito ruído. A primeira suspeita foi a alimentação fornecida pela USB, que ao medir com um multímetro, foi constatado que estava ligeiramente abaixo da tensão esperada ($\sim$ 4,8 V). Para descartar esse possível problema, o cabo USB de alimentação foi conectado a um carregador de celular com conexão USB e tensão medida de 5 V. Entretanto, ao realizar novamente o teste, o ruído persistiu, o que levou a desconfiar que a conexão de maneira direta era a causa do ruído.

Para utilizar um cabo serial *null modem* em vez de conexão direta, foi preciso encontrar um conector AT/Everex (Conector DB9 - *pin header*, comum em placa mãe antiga). Após retirar de um computador antigo, o cabo AT/Everex foi conectado na BeagleBoard e o cabo serial *null modem* foi conectado entre o AT/Everex e o adaptador RS-232-USB. Novamente o problema persistiu, deixando como possível causa o próprio adaptador RS-232-USB.

Como meio de substituir o adaptador, foi utilizado um computador Pentium-II que havia no

laboratório. O computador possuía como sistema operacional um Linux com *kernel* 2.4, além de ter porta serial e o programa *minicom*, o que permitiu a comunicação serial com sucesso.

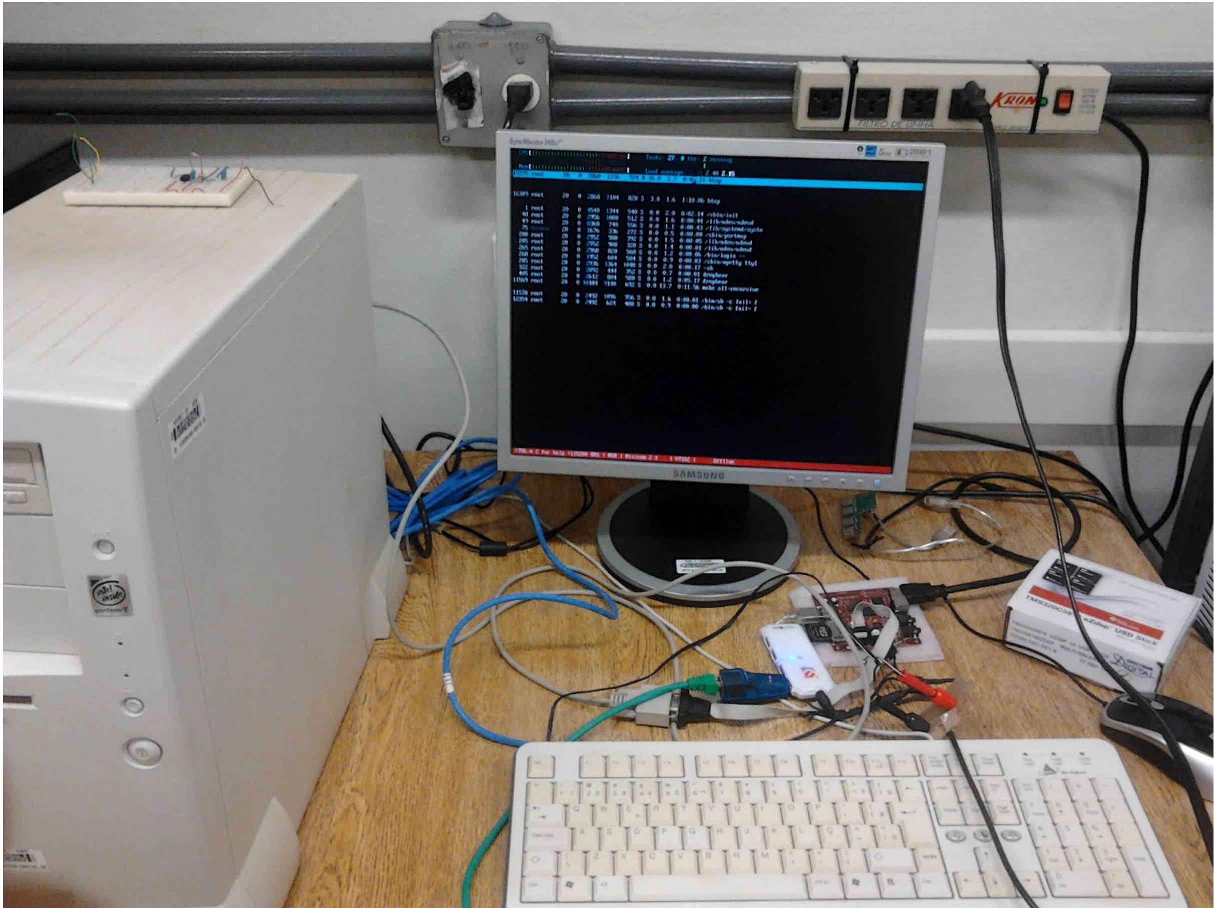


Figura 3.2: Ambiente utilizado para desenvolver o trabalho.

3.1.3 Rede

Para adicionar uma interface de rede a BeagleBoard, é necessário o uso de um adaptador USB-Ethernet. Como restrição elétrica da BeagleBoard, não é possível conectar periféricos que tenham consumo de energia considerável diretamente na porta USB *host* [1] da BeagleBoard. Visando resolver esse problema, um *hub* USB com alimentação externa foi usado para fornecer alimentação ao adaptador USB-Ethernet. O cabo Ethernet foi conectado diretamente ao *laptop* utilizado para o desenvolvimento.

[1]A revisão B da BeagleBoard tem a porta USB host desabilitada por padrão, sendo possível utilizar um cabo mini-USB para USB host no conector USB *On-The-Go*. Lembrando que é preciso curto circuitar o pino ID da USB *On-The-Go* para funcionar como USB host.

3.1.4 *Hardware* opcional

Como o intuito do trabalho é utilizar o poder de processamento da BeagleBoard, a utilização de interface gráfica ou mesmo o console através de um monitor é opcional. Para conectar um monitor é preciso um cabo HDMI-DVI e um monitor com entrada DVI.

O teclado e o mouse pode ser conectado de forma normal ao *hub* USB.

3.2 Criação de imagem para a BeagleBoard

O processo de criação da imagem de Linux para a BeagleBoard é apresentado nessa seção. A geração da imagem foi realizada utilizando a distribuição de Linux Fedora 16.

3.2.1 Pesquisa

Para criar uma imagem para a BeagleBoard, inicialmente foi feita uma pesquisa e testes iniciais sobre as alternativas disponíveis. Entre as alternativas, as que chamaram mais atenção foram Open Embedded, Yocto Project e a distribuição Ångström. Todas as opções usam de forma direta o Open Embedded para a geração de imagem. O Open Embedded inicialmente foi descartado pois a sua versão mais recente datava de março de 2011. O Yocto Project é suportado pela Linux Foundation e utiliza como *framework* de criação o Open Embedded, além de outros aplicativos auxiliares. O Yocto Project foi testado, mas devido a problemas para compilar os *drivers* [2] utilizados pelo DSP (possivelmente por falta de experiência), também foi deixado de lado.

Por ultimo, além da distribuição Ångström possuir um sistema de *release* contínuo, onde é possível sempre conseguir as versões mais recentes de todos os *softwares* utilizados, ela foi utilizada nos trabalhos anteriores na forma de imagens prontas e portanto testada com os módulos utilizados pelo DSP. A distribuição Ångström é na verdade um *layer* para o Open Embedded, que inclui imagens praticamente prontas para customização.

3.2.2 Ångström

A instalação da distribuição Ångström é relativamente simples, sendo necessário apenas resolver as dependências (as mesmas do Open Embedded) e então baixar uma versão recente

[2] O termo *driver* e módulo são utilizados de forma indistintas para referenciar o *software* utilizado para o funcionamento de diversos periféricos utilizados pelo microprocessador.

do repositório com controle de versões onde a distribuição é hospedada. Os passos de instalação estão no Anexo B.

Após a instalação da distribuição Ångström, foi compilada uma imagem padrão de console para a BeagleBoard a fim de testar a instalação. A primeira compilação por ser responsável também pela compilação do *cross compiler* para ARM, levou cerca de 4 horas [3] , mesmo com os pacotes de código fonte estando na máquina.

O passo de customização da imagem foi o que mais consumiu tempo do trabalho, pois houve a necessidade de estudar a *framework* Open Embedded, aprender como se faz um *layer*, criar *recipe*[4] do MPICH2 e testar a imagem gerada.

O *recipe* da imagem customizada foi feito utilizando o *recipe* de imagem padrão de console e então foram adicionados os pacotes dos módulos do DSP (*cmemk* e *dsplink*), do MPICH2 e do módulo do adaptador USB-Ethernet. Como não havia *recipe* do MPICH2 disponível no repositório do Ångström/Open Embedded, o pacote do MPICH2 foi criado. Os *recipes* criados são apresentados no Anexo C.

Para os primeiros testes, devido a disponibilidade inconstante de BeagleBoards, as imagens foram testadas no emulador QEMU. O emulador QEMU oficial não possui suporte a BeagleBoard, sendo necessário mudar o *target* da imagem gerada de BeagleBoard para ARM genérico para funcionar corretamente no QEMU. Devido a falta de semelhança com o *hardware* original, foi pesquisada alternativas. A pesquisa encontrou um *branch* [5] do QEMU desenvolvido pela organização Linaro. Nela é possivel emular uma BeagleBoard, com excessão de subsistemas como USB e DSP. Mesmo com a limitação do Linaro QEMU, o emulador permitiu testar as imagens geradas para a BeagleBoard, inclusive o SDK nativo. [6]

Com uma BeagleBoard em mãos, foi testado a imagem e a parte de comunicação de rede.

3.3 *Software* para o cluster

O software utilizado para testar o *cluster* é uma modificação do programa que calcula o π (Anexo A), com a diferença de que a parte de cálculo é separada em um arquivo fonte .c

[3]O computador utilizado foi um Intel (R) Core (TM) i3-2310M com 4Gb de memória RAM, e a geração de imagem foi feita com 4 *threads*.

[4]*Recipe* é o arquivo fonte com as instruções de como se gerar o pacote ao qual *recipe* se refere.

[5]Um *branch* é uma modificação de um *software* visando sanar problemas encontrados e que podem não ser interessantes para os mantedores do *software* em questão

[6]O *Software Development Kit* (SDK) é o conjunto de *software* (Como compilador e bibliotecas) utilizado para compilar programas. Posteriormente, mostrou-se desnecessário a utilizaçaão do SDK nativo, sendo necessário apenas o SDK no *host*.

diferente.

A separação da função de cálculo em um arquivo fonte (*critical.c*) diferente foi necessária para compilar as funções de cálculo a serem executadas no DSP. A geração de código para o DSP foi feita utilizando o *software* C6EZrun, especificamente o *front-end* C6RunLib. O C6RunLib permite compilar código para DSP e então gerar um biblioteca que pode ser *linkada* diretamente com o código gerado para a arquitetura ARM. Uma esquema explicativo pode ser observado na Figura 3.3.

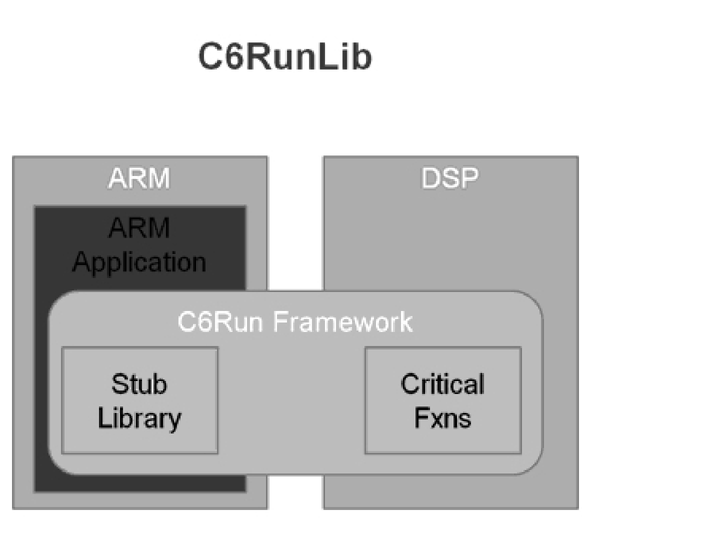


Figura 3.3: Funcionamento da ferramenta C6RunLib. [D-allred 2010]

O *software* C6EZrun foi baixado através do pacote *Digital Video Software Development Kit* (DVSDK) versão 4.01 que além do C6EZrun, engloba o compilador C6000 que é usado como *back-end* na geração de código para o DSP. O DVSDK tem como dependência o compilador para ARM CodeSourcery GCC 2009q1 (Um *branch* do GCC).

Para compilar a parte de ARM foi utilizado o mesmo compilador que é dependência do DVSDK. Para *linkar* (Juntar os arquivos compilados) o código fonte, como parâmetros foi passado o diretório das bibliotecas do MPICH2 compiladas pelo Open Embedded e as bibliotecas necessárias para a geração de código. O processo de geração dos executáveis foi automatizado com o uso de *shell scripting* (Anexo D. Foi gerado dois executáveis para a BeagleBoard e um para execução nativa no computador *host*. A Figura 3.4 mostra o processo de geração de código onde os cálculos são executados no DSP. O código com as funções de cálculo executada no ARM pode ser gerado compilando o arquivo fonte dos cálculos (*critical.c*) com o GCC ao invés do C6RunLib.

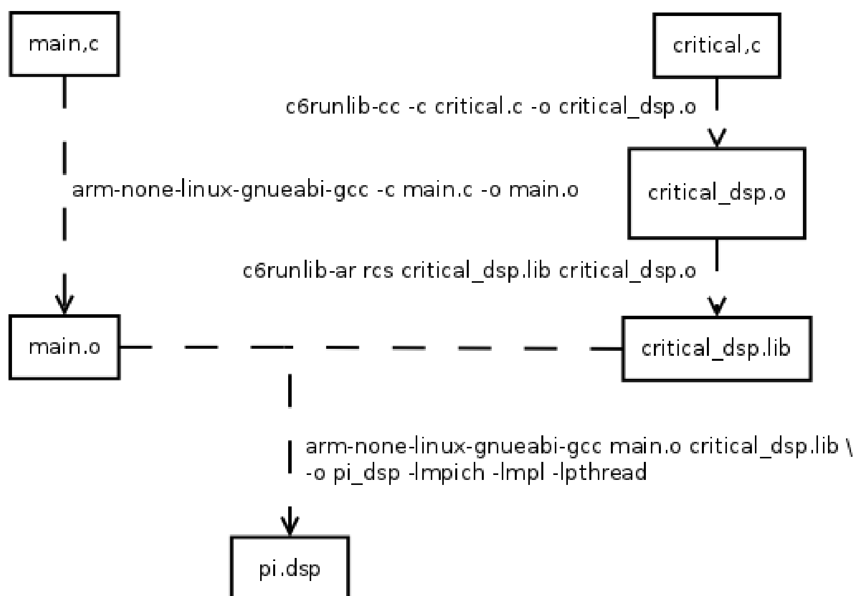


Figura 3.4: Geração de código com processamento feito no DSP.

3.4 Testes de funcionamento

Os testes de funcionamento foram feitos primeiramente utilizando o emulador QEMU e depois uma BeagleBoard revisão C, com 256 MBytes de RAM. No emulador QEMU foi testado o funcionamento correto da imagem e dos softwares gerados, inclusive do SDK nativo, utilizado para compilar como forma de teste o MPICH2.

Os testes na BeagleBoard foram feitos em diversas etapas. O item inicial a ser testado foi a Ethernet, com o intuito de fornecer acesso remoto por SSH a BeagleBoard. Com acesso remoto por SSH, foi transferido e testado, um programa simples para testar o funcionamento do DSP(programa exemplo de utilização do*dsplink*, incluso na imagem gerada), o binário gerado com os cálculos no ARM e por final o binário gerado com os cálculos no DSP.

3.5 Criação da página estilo *Wiki*

A criação da página *Wiki* foi feita documentando passo a passo todo o processo de criação de um *cluster* de DSP. Foi evitado ao máximo duplicar informações encontradas em outros lugares, optando sempre por colocar um *link* para o endereço eletrônico onde as informações pertinentes são disponibilizadas.

4 Resultados e Discussão

O teste do funcionamento da Ethernet apresentou um problema que foi logo identificado como a falta do módulo correspondente ao adaptador USB-Ethernet. A solução encontrada foi adicionar o módulo (*dm9601*) no *recipe* de geração da imagem.

A primeira tentativa do teste de funcionamento do DSP falhou devido a problemas com o endereço de memória RAM reservada para para o *buffer* do DSP. Após estudar o funcionamento do módulo *cmemk* (resposável por reservar *buffer* para o DSP), descobriu-se a necessidade de limitar a memória RAM disponível para o *Linux* através de um argumento passado para o *kernel*. A memória RAM do kernel foi limitada em 99 MBytes e o *cmemk* deixou reservado do endereço 0x87000000 ao endereço 0x8e000000, num total de 112 MBytes de RAM. Houve um pequeno desperdício de RAM, que poderia ser evitado com ajustes finos no parametro do *kernel* e dos endereços reservados pelo módulo *cmemk*, entretanto, como a natureza do aplicativo utilizado para o teste do *cluster* (cálculo do π) era dependente da CPU (*CPU bound*), optou-se por não realizar os ajustes finos. Com o problema de reserva de memória para o DSP resolvido, não houve problemas ao executar o programa simples de teste.

O binário gerado com os cálculos no ARM foi testado com o intuito de testar o funcionamento do MPICH2. Durante a execução do teste houve um erro relacionado com o caminho de execução do binário do MPICH2 responsável por gerenciar a execução dos processos do *cluster*. O problema foi resolvido criando um *link* simbólico dentro da pasta */usr/bin/* de *arm-angstrom-linux-gnueabi-mpiexec.hydra* para *mpiexec.hydra*.

Por fim, após ter testado a execução de todos os serviços necessários para o *cluster* de DSP funcionar, foi testado o binário com os cálculos no DSP. O programa rodou na BeagleBoard sem problemas. Para validar o *cluster*, devido a disponibilidade de apenas uma BeagleBoard, ouve a tentativa de criar um *cluster* heterogêneo entre o computador e a BeagleBoard que após muito trabalho em vão, resultou em uma pesquisa sobre a possibilidade de se criar um *cluster heterogêneo* com o MPICH2. Foi constatado que o MPICH2, diferentemente do MPICH1 [1] ,

[1] O MPICH1 foi utilizado nos trabalhos anteriores e permitiu a criação de *cluster* heterogêneo.

de fato não possui suporte a *clusters* heterogêneos.

Para cada teste com um dos binários que calcula o π, foram feitas 5 medidas e então tirada a média (O *script* utilizado para automatizar o teste está no Anexo E. No gráfico na Figura 4.1 pode ser observado os resultados obtidos.

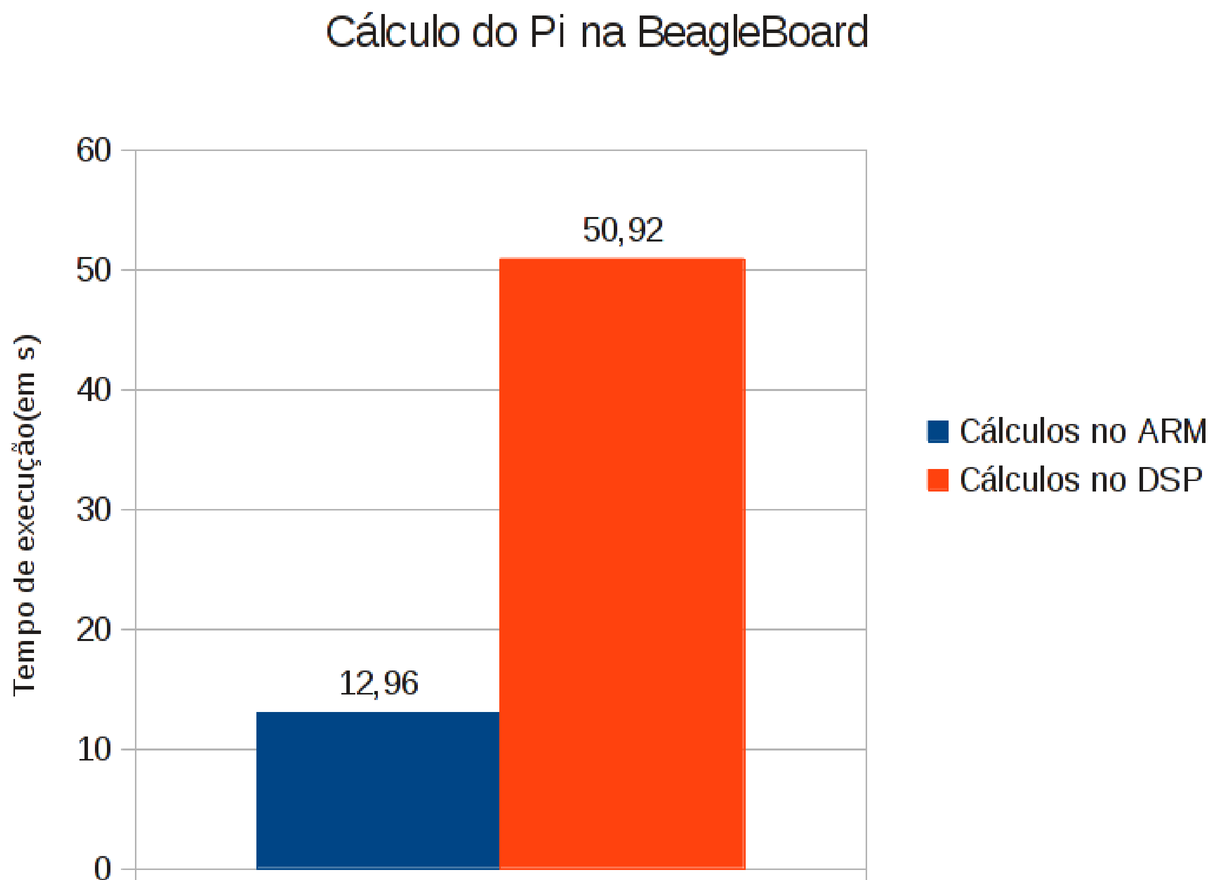


Figura 4.1: Resultados obtidos ao executar os testes.

A página no estilo *Wiki* foi primeiramente criado no *GitHub.com* com a finalidade de se conhecer a linguagem de marcação utilizada e posteriormente foi transferida para o *Wiki* conhecido entre os desenvolvedores de Linux embarcado *eLinux.org*.

5 *Conclusões e trabalhos futuros*

O teste com uma aplicação simples para o DSP foi oportuno, pois foi possível depurar o problema de reserva de RAM de maneira mais simples em relação a um teste utilizando o binário final com os cálculos no DSP.

O teste do binário com cálculos no ARM se mostrou eficiente para verificar o funcionamento do MPICH2 *cross* compilado. Foi interessante observar que a *linkagem* do binário ARM ocorreu tranquilamente utilizando versões diferentes do GCC, sendo o CodeSourcery GCC versão 4.3 utilizado para compilar o programa de cálculo do π e GCC 4.7 utilizado para a compilação cruzada do MPICH2.

O teste do binário com cálculos no DSP foi importante para mostrar o funcionamento correto do *cluster*. Infelizmente devido a disponibilidade de apenas uma BeagleBoard, foi impossível realizar o teste em um *cluster* de fato. Entretanto, assim como nos trabalhos anteriores o *cluster* utilizando MPICH funcionou de maneira correta, é praticamente certo o funcionamento com o MPICH2, visto que o todo o software para *cluster* funcionou com um nó.

A diferença de desempenho entre os binários com cálculo no DSP e no ARM é facilmente explicado pelo fato do núcleo DSP utilizado ser de ponto fixo, e os cálculos de ponto flutuante serem feitos via emulação por *software*. A emulação via *software* apresenta um elevadíssimo *overhead* no desempenho da aplicação, visto que o programa utilizado utiliza apenas ponto flutuante para os cálculos.

A criação de uma página no *Wiki* foi ligeiramente desafiadora devido a necessidade de se criar um material de qualidade principalmente pela exposição gerada pela *internet*.

Entre os principais desafios enfrentados durante o projeto estão a utilização da *framework* Open Embedded, a atualização do *kernel* e do MPICH para versões mais recentes (*kernel* 3.2 e MPICH2 1.5b1) e a criação de uma página de *Wiki*.

O projeto de *cluster* de DSP se mostrou muito interessante, inclusive pelo caráter inédito da implementação utilizando MPI. Ao pesquisar na literatura científica, não foi encontrada uma

implementação com as características de utilizar Linux, MPI e DSP. O único tipo de literatura científica encontrada a respeito de *cluster* de DSP é o realizado por meio de *hardware*.

A variedade de tecnologias utilizadas para no *cluster* de DSP é vasta, proporcionando um ambiente rico para um aprendizado em distintas áreas.

Uma continuação interessante para o trabalho seria o desenvolvimento no sentido de criar um *Grid* de celulares utilizando o DSP para processamento.

Bibliografia

[Araújo, Dourado e Maciel 2010]ARAÚJO, S. A.; DOURADO, J. R.; MACIEL, C. D. Low-power cluster using omap3530. In: *Fourth European DSP in Education and Research Conference*. [S.l.: s.n.], 2010.

[ARM Limited 2011]ARM LIMITED. *ARM Architecture Reference Manual*. 2011.

[Baker e Sterling 2000]BAKER, M.; STERLING, T. *Cluster Computing White Paper*. [S.l.], 2000.

[Carol Atack e Alex van Someren 1993]Carol Atack; Alex van Someren. *The ARM RISC Chip: A Programmer's Guide*. [s.n.], 1993. Disponível em: <http://www.ot1.com/arm/armchap1.html>.

[Coley 2012]COLEY, G. *BeagleBoard System Reference Manual*. 2012. URL: http://beagleboard.org/static/BBSRM_latest.pdf. Last Visited: 09/04/2012.

[D-allred 2010]D-ALLRED. *C6RunLib.gif*. 2010. Disponível em: <http://processors.wiki.ti.com/index.php/File:C6RunLib.gif>.

[Dourado e Maciel 2011]DOURADO, J. R.; MACIEL, C. D. Low-power eembedded arm+dsp cluster. In: *Conferência Brasileira de Sistemas Embarcados Críticos*. [S.l.: s.n.], 2011.

[Electronic Industries Association. Engineering Dept 1969]ELECTRONIC INDUSTRIES ASSOCIATION. ENGINEERING DEPT. *Interface between data terminal equipment and data communication equipment employing serial binary data interchange*. Electronic Industries Association, Engineering Dept., 1969. (EIA standard). Disponível em: <http://books.google.com.br/books?id=Tp08AAAAIAAJ>.

[Foster 2002]FOSTER, I. What is the grid? a three point checklist. *Grid Today*, v. 1, n. 6, p. 22?25, 2002.

[GNU 2012]GNU. *Cross-Compilation*. 2012. Last Visited: 11/06/2012. Disponível em: <http://www.gnu.org/savannah-checkouts/gnu/automake/manual/html_node/Cross_002dCompilation.html>.

[IEEE Milestones]IEEE MILESTONES. *Speak Spell, the First Use of a Digital Signal Processing IC for Speech Generation, 1978*. Last Visited: 11/06/2012. Disponível em: <http://www.ieeeghn.org/wiki/index.php/Milestones:Speak_

[Koninklijke Bibliotheek]KONINKLIJKE BIBLIOTHEEK. *What is emulation?* Last Visited: 29/06/2012. Disponível em: <http://www.kb.nl/hrd/dd/dd_projecten/projecten_emulatiewatis-en.html>.

[Metcalfe e Boggs 1976]METCALFE, R. M.; BOGGS, D. R. Ethernet: distributed packet switching for local computer networks. *Commun. ACM*, ACM, New York, NY, USA, v. 19, n. 7, p. 395–404, jul. 1976. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/360248.360253>.

[MOTOROLA]MOTOROLA. *DROID 2*. URL: http://developer.motorola.com/products/droid2/. Last visited: 10/06/2012.

[Sedra e Smith 2009]SEDRA, A.; SMITH, K. *Microelectronic Circuits*. [S.l.]: Oxford University Press, 2009. (Oxford Series in Electrical and Computer Engineering). ISBN 9780195323030.

[Smith 1998]SMITH, R. D. Simulation article. 1998. Disponível em: <http://www.modelbenders.com/encyclopedia/encyclopedia.html>.

[Snir et al. 1998]SNIR, M. et al. *MPI-The Complete Reference, Volume 1: The MPI Core*. 2nd. (revised). ed. Cambridge, MA, USA: MIT Press, 1998. ISBN 0262692155.

[Stallings 2008]STALLINGS, W. *Operating Systems: Internals and Design Principles*. 6th. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2008. ISBN 0136006329, 9780136006329.

[Texas Instruments 2009]TEXAS INSTRUMENTS. *OMAP3530 Datasheet*. 2009.

[Texas Instruments 2010]TEXAS INSTRUMENTS. *TMS320C64x/C64x+ DSP CPU and Instruction Set*. 2010.

[Timothy Prickett Morgan 2011]Timothy Prickett Morgan. *ARM Holdings eager for PC and server expansion*. [s.n.], 2011. Disponível em: <http://www.theregister.co.uk/2011/02/01/arm_holdings_q4_2010_numbers/>.

[Torvalds e Diamond 2001]TORVALDS, L.; DIAMOND, D. *Just for Fun: The Story of an Accidental Revolutionary*. [S.l.]: HarperInformation, 2001. ISBN 0066620724.

[USB IMPLEMENTERS 2012]USB IMPLEMENTERS. *USB-OTG reference*. 2012. URL: http://www.usb.org/developers/onthego/. Last Visited: 09/04/2012.

[Vikketorr 2010]VIKKETORR. *Kernel-microkernel.svg*. 2010. URL: http://en.wikipedia.org/wiki/File:Kernel-microkernel.svg.

[Ylonen e Lonvick 2006]YLONEN, T.; LONVICK, C. *The Secure Shell (SSH) Authentication Protocol*. IETF, January 2006. RFC 4252 (Proposed Standard). (Request for Comments, 4252). Disponível em: <http://www.ietf.org/rfc/rfc4252.txt>.

[Ysangkok 2007]YSANGKOK. *Kernel-simple.png*. 2007. Http://en.wikipedia.org/wiki/File:Kernel-simple.png.

ANEXO A – Cálculo do π com o MPI

Retirado dos exemplos do MPICH2, créditos para o Argonne National Laboratory.

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f(double a)
{
    return (4.0 / (1.0 + a*a));
}

int main(int argc,char *argv[])
{
    int    n, myid, numprocs, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int    namelen;
    char   processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);

    fprintf(stdout,"Process %d of %d is on %s\n", myid, numprocs,
        processor_name);
    fflush(stdout);

    n = 10000;            /* default # of rectangles */
    if (myid == 0)
    startwtime = MPI_Wtime();

    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

```
    h   = 1.0 / (double) n;
    sum = 0.0;

    /* A slightly better approach starts from large i and works back */
    for (i = myid + 1; i <= n; i += numprocs)
    {
        x = h * ((double)i - 0.5);
        sum += f(x);
    }
    mypi = h * sum;

    MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

    if (myid == 0) {
        endwtime = MPI_Wtime();
        printf("pi is approximately %.16f, Error is %.16f\n", pi, fabs(pi -
            PI25DT));
        printf("wall clock time = %f\n", endwtime-startwtime);
        fflush(stdout);
    }

    MPI_Finalize();
    return 0;
}
```

ANEXO B – Comandos para instalar o Ångström

Comandos para instalar o Open Embedded com o *layer* Ångström e suas dependências.

```
sudo yum groupinstall 'Development Tools'
sudo yum update
sudo yum install python m4 make wget curl ftp cvs subversion tar bzip2\
 gzip unzip python-psyco perl texinfo texi2html diffstat openjade\
 docbook-style-dsssl docbook-style-xsl docbook-dtds docbook-utils sed\
 bison bc glibc-devel glibc-static gcc binutils pcre pcre-devel git\
 quilt groff linuxdoc-tools patch linuxdoc-tools gcc-c++ help2man\
 perl-ExtUtils-MakeMaker tcl-devel gettext ccache chrpath cmake ncurses\
 apr
git clone git://github.com/Angstrom-distribution/setup-scripts.git
```

ANEXO C – Recipes desenvolvidos durante o projeto

São apresentados os *recipes* do MPICH2 e da customização da imagem de console com todo o *software* necessário para a criação de um *cluster* de BeagleBoard.

MPICH2:

```
DESCRIPTION = "MPICH2 is a high-performance and widely portable
    implementation of the Message Passing Interface (MPI) standard (both MPI
    -1 and MPI-2). "
HOMEPAGE = "http://www.mcs.anl.gov/research/projects/mpich2/index.php"
SECTION = "console/scientific"

PROVIDES = "mpi"

RDEPENDS = "ssh"

EXTRA_OECONF = "--with-device=ch3:sock --disable-f77 --disable-fc"

SRC_URI = "http://www.mcs.anl.gov/research/projects/mpich2/downloads/
    tarballs/${PV}/${P}.tar.gz;md5sum=8be7521600b69b18b9804bd842c661e0"

inherit autotools pkgconfig

do_configure() {
    oe_runconf
}

do_install () {
    mkdir -p "${D}/usr/bin"
    mkdir "${D}/etc"
    mkdir -p "${D}/usr/share/man/man4"
    autotools_do_install
}
```

Imagem customizada:

```
#Angstrom bootstrap image
require ${TOPDIR}/sources/meta-angstrom/recipes-images/angstrom/console-
    image.bb

LICENSE = "GPL"

DEPENDS += "ti-linuxutils \
            mpich2 \
            "

IMAGE_INSTALL += "ti-cmem-module \
                  ti-lpm-module\
                  ti-sdma-module\
                  ti-dsplink-module\
                  ti-dsplink-examples\
                  ti-codec-engine-examples\
                  kernel-module-dm9601\
                  mpich2 \
                  task-native-sdk \
                  coreutils\
                  "

export IMAGE_BASENAME = "bb-cluster-image"
```

ANEXO D – Shell script usado para compilar o binário final

O *shell script* utilizado para compilar o binário compila para a BeagleBoard as versões utilizando o ARM e o DSP para os cálculos e uma versão *x86* apenas para testes.

```
#/bin/bash
#  Redistribution and use in source and binary forms, with or without
#  modification, are permitted provided that the following conditions are
#  met:
#
#  * Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
#  * Redistributions in binary form must reproduce the above
#    copyright notice, this list of conditions and the following disclaimer
#    in the documentation and/or other materials provided with the
#    distribution.
#  * Neither the name of the  nor the names of its
#    contributors may be used to endorse or promote products derived from
#    this software without specific prior written permission.
#
#  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
#  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
#  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
#  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
#  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
#  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
#  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
#  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
#  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
#  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
#  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#
```

```
critical_source="critical.c"
main_source="main.c"

base_path="/home/tcc/another/setup-scripts/build/tmp-angstrom_v2012_05-
    eglibc/work/armv7a-angstrom-linux-gnueabi/mpich2-1.5a2-r0/package"

echo "Building DSP critical"
./c6runlib-cc -c $critical_source -o critical_dsp.o
./c6runlib-ar rcs critical_dsp.lib critical_dsp.o

echo "Building ARM critical"
arm-none-linux-gnueabi-gcc -c $critical_source -o critical_arm.o -
    I$base_path/usr/include/

echo "Building main (always on ARM)"
arm-none-linux-gnueabi-gcc -c $main_source -o main.o -I$base_path/usr/
    include/

echo "Linking main.o with DSP critical"
arm-none-linux-gnueabi-gcc -I$base_path/usr/include/ -L$base_path/usr/lib/
    main.o critical_dsp.lib -o pi_dsp -lmpich -lmpl -lpthread

echo "Linking main.o with ARM critical"
arm-none-linux-gnueabi-gcc -I$base_path/usr/include/ -L$base_path/usr/lib/
    main.o critical_arm.o -o pi_arm -lmpich -lmpl

echo "Building x86 binary"
gcc main.c critical.c -o pi_x86 -lmpich
```

ANEXO E – Shell script usado para realizar os testes

Ao executar o *shell script*, o resultado é armazenado nos arquivos results_arm e results_dsp.

```
#!/bin/bash
#  Redistribution and use in source and binary forms, with or without
#  modification, are permitted provided that the following conditions are
#  met:
#
#  * Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
#  * Redistributions in binary form must reproduce the above
#    copyright notice, this list of conditions and the following disclaimer
#    in the documentation and/or other materials provided with the
#    distribution.
#  * Neither the name of the  nor the names of its
#    contributors may be used to endorse or promote products derived from
#    this software without specific prior written permission.
#
#  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
#  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
#  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
#  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
#  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
#  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
#  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
#  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
#  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
#  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
#  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#

echo "Starting tests"
for i in {1..5..1}
  do
     echo "ARM test number $i"
```

```
    echo "mpiexec.hydra -f hosts ./pi_arm tee - >> results_arm"
    mpiexec.hydra -f hosts ./pi_arm tee - >> results_arm
done

for i in {1..5..1}
  do
    echo "DSP test number $i"
    echo "mpiexec.hydra -f hosts ./pi_dsp tee - >> results_dsp"
    mpiexec.hydra -f hosts ./pi_dsp tee - >> results_dsp
done
```