

SAUL ARAÚJO ANDRADE

**IMPLEMENTAÇÃO E AVALIAÇÃO DE
UM CLUSTER DE BEAGLEBOARDS
AMBIENTADAS EM LINUX
UTILIZANDO MPI**

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

Curso de Engenharia de Computação

ORIENTADOR: Professor Doutor Carlos Dias Maciel

São Carlos

2010

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

A553i Andrade, Saul Araújo
Implementação e avaliação de um *cluster* de
beagleboards ambientadas em LINUX utilizando MPI / Saul
Araújo Andrade ; orientador Carlos Dias Maciel. -- São
Carlos, 2010.

Trabalho de Conclusão de Curso (Graduação em
Engenharia de Computação) -- Escola de Engenharia de São
Carlos da Universidade de São Paulo, 2010.

1. Microprocessadores. 2. Beagle. 3. OMAP.
4. Cluster. 5. MPI. 6. ARM. I. Título.

FOLHA DE APROVAÇÃO

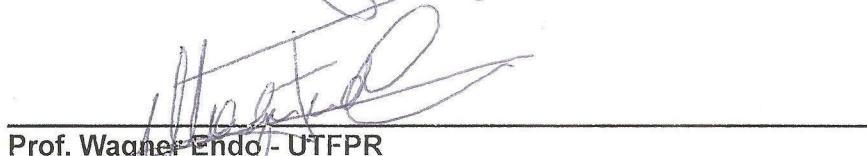
Nome: Saul Araújo Andrade

Título: "Implementação e Avaliação de um Cluster de Beagle Boards Ambientadas em Linux Utilizando MPI"

Trabalho de Conclusão de Curso defendido e aprovado
em 16 / 06 / 2010,

com NOTA 9,3 (Nove, Três), pela comissão julgadora:


Prof. Assistente Carlos Goldenberg - SEL/EESC/USP


Prof. Wagner Endo - UTFPR


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

Aos meus pais, Claudionor Medeiros de Andrade e Maria Auxiliadora Araújo Andrade, pela paciência que tiveram e continuam a ter comigo, e por acreditarem que eu conseguiria vencer esta fase da vida mesmo estando tão longe deles...

E aos meus irmãos, José Ramos de Araújo Neto e Lorena Araújo Andrade, por se preocuparem comigo e terem lembrado de me ligar em todos os meus aniversários quando não puderam estar comigo. Este trabalho é dedicado a vocês.

Agradeço a todos os que me apoiaram desde o início do curso: aos amigos que sempre tive e aos novos que conheci, e que por diversas vezes me deram força para continuar (espero tê-los ajudado de alguma forma também); ao meu orientador Prof. Dr. Carlos Dias Maciel, pela incrível confiança que depositou em mim durante este projeto e, em especial; à Giovana Milanetto, que se tornou uma pessoa muito importante na minha vida e nunca deixou de insistir para que eu dormisse mais de 3 horas por noite...

À todos vocês, muito obrigado.

Sumário

1	Introdução	9
1.1	Objetivos	10
2	Fundamentação Teórica	11
2.1	Arquitetura de Computadores	11
2.1.1	Beagle Board e a Arquitetura ARMV7-A	12
2.2	Sobre Sistemas Operacionais e Outros Conceitos	19
2.2.1	Kernel	19
2.2.2	Sistemas de arquivos	20
2.3	<i>Cluster</i>	21
2.3.1	MPI e MPICH	23
3	Métodos	24
3.1	Kit de energia (Fonte de Alimentação)	25
3.2	Distribuição LINUX Utilizada	26
3.3	Plataforma de Trabalho	27
3.3.1	Confecção dos Cabos Seriais	27
3.3.2	Processo de <i>BOOT</i>	28
3.4	Ajuste Fino	30
3.4.1	Utilizando o OpenEmbedded (OE)	30
3.4.2	Compilando o Kernel e Drivers Separadamente	31
3.5	Escolha da Implementação MPI e Montagem do Cluster	32
3.6	Resultados	33
3.6.1	Cluster Heterogêneo	34
4	Conclusões	36
Referências Bibliográficas		52

Listas de Figuras

2.1	A clássica arquitetura de Von Neumann - uma CPU e uma unidade de memória principal interligadas por um caminho único	12
2.2	Beagle Board e suas dimensões inacreditavelmente reduzidas: 3" x 3"	13
2.3	RS-232 conector DB-25 apresentando na totalidade os sinais de comunicação definidos pelo padrão em questão	15
2.4	Cabo Ethernet RJ-45 e Modelo de camadas OSI	16
2.5	USB-OTG	17
2.6	<i>Cross-Compiling.</i> Geração de binários através de um compilador instalado em uma máquina de arquitetura diferente da arquitetura alvo.	18
2.7	Kernel - camada de abstração mais baixa que provê acesso aos recursos de hardware mais essenciais para aplicações de mais alto nível.	19
2.8	Cluster da Universidade de Boise, claramente um cluster <i>local</i>	22
2.9	Exemplo de Grid com computadores espalhados ao redor do mundo trabalhando juntos.	22
3.1	Regiões da Beagle Board trabalhadas durante este projeto	24
3.2	kit de energia e seus componentes	25
3.3	Esquema de montagem do kit de energia ilustrando como devem estar dispostos os componentes	25
3.4	Cartão de memória onde o sistema de arquivos será acomodado	26
3.5	Cabo serial montado pelo aluno para primeira interação com a BB	27
3.6	HUB USB com conector trocado para o padrão miniUSB	28
3.7	<i>Beagle Board</i> conectada ao computador do laboratório via o cabo serial montado pelo aluno.	29
3.8	Imagen do software minicom exibindo o console serial da BB executando o u-boot	29
3.9	<i>Beagle Board</i> rodando o gerenciador gráfico enlightenment	30
3.10	Adaptador USB/Ethernet com <i>chipset</i> correspondente ao módulo linux dm9601	31
3.11	HUB Ethernet com elevada taxa de comunicação. Utilizado para amenizar a latência da rede.	33
3.12	<i>Kit</i> DSP L137 para o qual o MPICH foi portado também.	33

3.13	Plataforma de trabalho em sua forma final	34
3.14	Cluster Heterogêneo formado pelo computador do laboratório, duas BB's e dois <i>kits DSP</i>	35
3.15	Imagen do MPI executando o programa de cálculo paralelo de PI	35
4.1	Código fonte utilizado nos testes dos <i>cluster's</i>	44
4.2	Configurando o <i>minicom</i> para acesso à BB	47
4.3	Tela da BB já no segundo estágio de inicialização (u-boot)	48
4.4	Aplicação do CI LM7805 como regulador de tensão.	51

Lista de Tabelas

2.1	ARM Cortex e Algumas de suas aplicações.	14
4.1	Especificações da Beagle Board e do OMAP3530	37
4.2	Evolução dos processadores ARM	38

Resumo

Quando se fala em supercomputação, uma abordagem muito comum é dada pelos ditos *Cluster*. Um *cluster*, nada mais é que a associação de vários computadores processando um mesmo problema. Apesar de parecer a solução mais lógica para a resolução de grandes questões, imagine um problema tão complexo que exija 100 computadores para ser resolvido dentro de um tempo razoável. Cada computador gasta uma determinada quantia de energia, e a energia de 100 computadores é aceitável. No entanto, se o número de computadores simplesmente crescer com a complexidade do problema a ser resolvido, um limite será eventualmente atingido.

Este trabalho tem como intenção otimizar o uso da energia disponível, propondo um *cluster* constituído por OMAP's, um novo micro-processador de baixo consumo desenvolvido pela Texas Instruments detentor de uma capacidade de processamento significativa. Para tanto, a *Beagle Board* (BB), uma plataforma ARM que pode ser alimentada via USB, foi escolhida como apropriada para o ínicio dos experimentos. Após algum esforço para se obter uma plataforma de trabalho operacional (instalação do linux e recompilação de alguns drivers), era hora de decidir como gerenciar o cluster de fato. O MPICH, uma implementação do famoso protocolo de passagem de mensagens atendia a todas as necessidades do projeto. O último obstáculo a ser transposto seria compilar este software para a arquitetura da BB, recentemente realizado.

Como resultado, o cluster de BB's provou-se viável.

Palavras-Chave: Beagle, OMAP, Cluster, MPI, Supercomputação, Multiprocessamento

Abstract

When it comes to the field of supercomputing a very common approach is a Cluster, many computers associated to solve a single problem sounds very good to begin with, however, imagine a problem so complex that it takes 100 computers to be solved within a reasonable time, needless to say these computers spend energy, hence, the more computers are added to the cluster the more energy is given away. The Question is: “Should more computers be continuosly added to a cluster to keep solving more complex problems?”. Assuming there is no unlimited power source avaiable, a practical limit will eventually be reached.

This work intends to take the most of the current avaiable power by proposing a cluster made of OMAP's, a new low power micro-processor developed by Texas Instruments capable of significant accomplishments. For this purpose, The Beagle Board (BB), a usb powered ARM platform, was chosen as a suitable platform to begin experimenting on the OMAP. After some effort to get it working (booting linux and tweaking some missing drivers) it was time to decide how to manage the cluster. MPICH attended to all the needs of the project. The last matter to be overcome was cross-compiling MPICH, which has been recently achieved.

The outcome reflected all the expectations that surrounded it, a BB cluster has been proved viable.

Keywords: Beagle, OMAP, Cluster, MPI, Supercomputing, Multiprocessing

Capítulo 1

Introdução

A curiosidade é uma característica humana intrínseca, e traz consigo a vontade de investigar. Através de uma observação do que o cerca, o ser humano é impelido a formular os mais variados questionamentos, de natureza prática ou filosófica. Aliado ao senso de finitude (proveniente de outra característica inerente à humanidade, a mortalidade), a curiosidade move o homem; integrando como peça importante o motor de idéias tão presente nesta espécie. E o que seria a tecnologia senão a formalização deste motor?

Através da tecnologia, questões são respondidas, de fato, mas também, novas questões são geradas. Em um dos ramos tributários da tecnologia, mais especificamente, na área da computação, métodos para se resolver problemas complexos são constantemente discutidos.

Neste trabalho, foi realizada uma incursão pela área da supercomputação, e investigou-se a viabilidade de um *cluster* constituído por plataformas ARM. Para tanto, escolheu-se uma plataforma chamada *Beagle Board* (BB), construída a partir de um processador da família OMAP3; designou-se a distribuição *Angstrom Linux* para ser instalada e carregada nas placas através de um cartão de memória SD e, adaptou-se os periféricos necessários como teclado e mouse para funcionarem a partir de um hub USB conectado as portas USB-OTG das BB's. Como primeira interface com as placas, uma conexão serial foi estabelecida com um computador do laboratório, e as variáveis de ambiente necessárias ao boot e à exibição do vídeo das BB's via saída digital foram alteradas com sucesso (o *software* utilizado para a comunicação serial foi o *minicom*). Também foram montados *kit's* de energia que consistem de fontes de 12V DC, capazes de suprir até 1A de corrente; CI's LM7805, para que 2 saídas de 5V DC estejam disponíveis (uma para a BB e outra para o hub USB com alimentação externa em cada *kit*) e um *cooler* onde os reguladores de tensão estão fixados para uma eficiente dissipação de calor (cada *kit* tem capacidade de suprir uma BB).

Só então, com as plataformas de trabalho montadas e sendo capazes de executar em modo *stand-alone*, o MPICH teve seu código traduzido para a arquitetura ARM. As duas formas de tradução foram testadas: através de *cross-compiling* e compilação nativa, dado o elevado clock do OMAP3530 - 720 MHz.

Por fim, adquiriu-se 2 adaptadores USB/Ethernet, um *switch* ethernet de boa qualidade e as BB's puderam ser dispostas em forma de *cluster* (utilizando o MPICH). Alguns testes foram realizados, porém, devido a problemas com o driver desta interface USB/ethernet específica, não foi possível obter medidas para caracterizar a qualidade do arranjo montado.

No entanto, havia no laboratório 2 kits de desenvolvimento (de arquiteturas diferentes da BB) que possuíam saídas ethernet nativas; através de *cross-compiling* foi possível traduzir o MPICH também para estes dois kits e operá-los em conjunto com as BB's, caracterizando um cluster heterogêneo e provando assim o potencial desta técnica.

1.1 Objetivos

Buscou-se neste trabalho, investigar a possibilidade de se montar um *cluster*, cujos nós fossem dados por plataformas *Beagle Board*. E através disto obter um conhecimento mais aprofundado de arquiteturas ARM (arquitetura da *Beagle Board*) e o novo processador OMAP3530; linux embarcado (tanto customização de kernel quanto compilação de drivers) e protocolos de comunicação (Serial, USB, USB-OTG, entre outros); adquiriu-se também conhecimentos de como realizar compilação cruzada (*cross-compiling*) bem como da tecnologia que empregada para gerenciar o *cluster* em si, MPICH. Dentre os principais desafios estiveram:

- A customização do kernel linux para se adaptar à BB
- Confecção de cabos USB e Serial RS232
- Adaptação nas portas USB-OTG da *Beagle Board* para induzir a função de principal (*host*)
- Adaptações realizadas em HUB's USB e hardwares que necessitassem de alimentação externa para que fossem conectados ao *kit* de energia
- Mudar o arquivo *configure* do MPICH para que permitisse *cross-compiling*
- Fazer o *cross-compiling* do driver do adaptador USB/Ethernet a partir do seu código fonte

Seria omissão deixar de citar que trabalhar em uma máquina diferente das usuais tem muitos inconvenientes em relação às usuais porém, está ocorrendo uma grande mobilização para promover este tipo de esforço no meio *Open-Source*, com destaque o time de desenvolvimento do *Angstrom-Linux* que mantém um repositório atualizado de pacotes já compilados para diferentes arquiteturas; além de um gerador de imagens linux customizáveis online (*narcissus*).

Capítulo 2

Fundamentação Teórica

A seguir, para a melhor compreensão deste trabalho, os conceitos mais importantes introduzidos anteriormente são descritos. Espera-se que ao final deste capítulo, a idéia principal por trás dos mesmos esteja clarificada de maneira satisfatória.

2.1 Arquitetura de Computadores

Design de computadores é a arte de produzir um computador de boa performance para um dado conjunto de especificações a um baixo custo. Arquitetura de computadores é a arte de criar um conjunto de especificações que perdurará por gerações de tecnologia. (Robert J. B. - Universidade de Iowa)

Arquitetura de computadores é o projeto do computador por completo, incluindo seu conjunto de instruções (*ISA - Instruction-Set Architecture*) e componentes de hardware (*HSA - Hardware-System Architecture*). Um computador é geralmente visto em termos de seu ISA, que diz respeito a como os programadores de linguagem de máquina interagem com o computador. Em contraste, o HSA lida com os sistemas maiores de hardware, incluindo CPU (central processing unit, inglês para unidade central de processamento), sistema de armazenamento de dados e sistema de entrada e saída (I/O - Input/Output)[3]

É importante citar que computadores de mesmo ISA geralmente terão a habilidade de executar os mesmos programas, isto leva diretamente à noção de família de computadores. Em suma, família de computadores nada mais é que um conjunto de implementações para o mesmo ISA. Na figura 2.1, é possível observar a arquitetura de Von Neumann (VN) que apresenta uma CPU, um sistema de memória principal e sistema de I/O. Nas máquinas VN, instruções são carregadas sequencialmente e a CPU executa uma operação de cada vez; existe apenas um caminho entre o sistema de memória principal e a unidade de controle da CPU; o sistema de memória principal detém o programa que controla toda a operação do computador, e este pode manipular seu próprio programa mais ou menos da mesma forma que qualquer outro dado na memória[3]

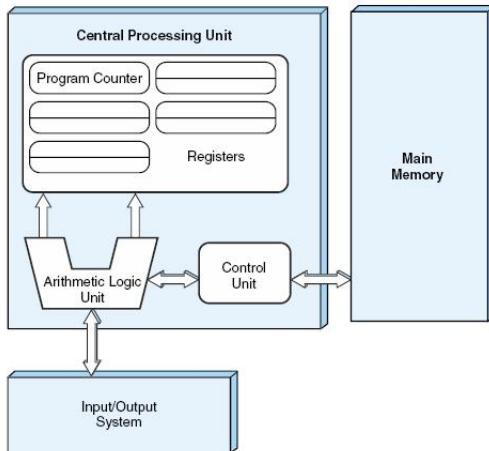


Figura 2.1: A clássica arquitetura de Von Neumann - uma CPU e uma unidade de memória principal interligadas por um caminho único

A noção de arquitetura de computadores é de suma importância para a percepção do potencial que este trabalho carrega, visto que a realização mais significativa deste foi fugir da arquitetura do PC convencional para uma outra cujo ISA é bem mais simples e o processador muito mais econômico porém, cujo processamento atingiu um patamar significativo e tende a crescer ainda mais; trata-se do OMAP3530 que será melhor detalhado a seguir.

2.1.1 Beagle Board e a Arquitetura ARMV7-A

Para o único dos experimentos com o OMAP 3530 uma plataforma chamada *Beagle Board* (BB) foi eleita. A BB é um mini-computador de baixíssimo consumo (comparável ao de um celular), com processador da família OMAP3 de arquitetura ARMv7-A (Seção 2.1), otimizado para ambiente Linux, que pode ser alimentado via USB (*universal serial bus*, Seção 2.1.1.3) com baixa dissipação de potência. A família OMAP3 [12] é formada por processadores de alto-desempenho projetados para garantir processamento gráfico de qualidade. Alguns de seus principais subsistemas são: microprocessador ARM CortexTM-A8 720MHz (ver tabela 4.2 nos anexos), acelerador gráfico 2D/3D capaz de renderizar 10 milhões de polígonos por segundo e DSP (Digital Signal Processor, inglês para processador digital de sinais¹) com core C64x+ 430MHz, tudo isso garante à BB uma performance comparável a de um laptop com um consumo de pico de 2W. Na figura 2.2, pode-se ter uma idéia da aparência física da BB, enquanto que na tabela 4.1 (ver Anexo A), as características do OMAP3530 já citadas anteriormente são acrescidas de mais detalhes.

¹Os DSP's são processadores especiais que executam algumas operações de maneira otimizada, tendo como propósito o tratamento de sinais como áudio e vídeo.

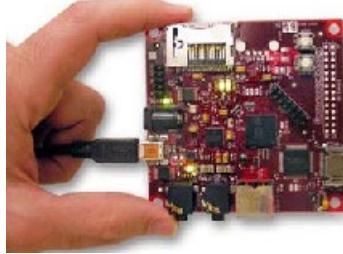


Figura 2.2: Beagle Board e suas dimensões inacreditavelmente reduzidas: 3" x 3"

A BB mede apenas 3" x 3" e possui saída de vídeo digital. A corrente máxima drenada pela plataforma sozinha é de 350mA, sendo que sua tensão de alimentação é de 5V (padrão USB, ver 2.1.1.3), o que a torna um excelente ponto de partida para uma nova geração de portáteis de baixo consumo e portanto, maior autonomia.

Quanto ao processamento de sinais, são infundáveis as aplicações médicas que o OMAP35x poderia oferecer, um exemplo seria aumentar o grau de inteligência de algoritmos para detecção de QRS durante um eletrocardiograma, garantindo ao médico que operasse tal aparelho uma maior confiabilidade em suas leituras. Vale lembrar que este não é o estágio final de desenvolvimento desta família de processadores, a Texas Instruments anunciou recentemente que sua nova versão poderá superar 1GHz de clock, mantendo o baixo consumo.

Sobre a arquitetura do OMAP (ARMV7-A), ARM é uma arquitetura de 32-bit com um conjunto reduzido de instruções (*ISA - Instruction Set Architecture*) concebida originalmente pela empresa Acorn Computers. Antes da sigla mais moderna, a arquitetura ARM era conhecida como *Acorn RISC (Reduced Instruction Set Computer) Machine*, e só mais tarde, *Advanced RISC Machine*². A máquina avançada de instruções reduzidas foi projetada como uma arquitetura para *PC's*, um mercado agora dominado pela família x86 utilizada pela IBM; no entanto, a relativa simplicidade dos processadores ARM tornou-os apropriados para aplicações *low-power*, isto fez desta família a dominante no mercado de tecnologias móveis e embarcadas.

A arquitetura ARM é licenciável e portanto, o desenvolvimento de processadores deste tipo pode ser feito por outras empresas além da *ARM Holdings*, o que se mostrou bastante eficaz como tática de popularização da tecnologia. Algumas das empresas que têm ou já tiveram esta licença incluem: *Atmel, Broadcom, Freescale, Intel, LG, Marvell Technology Group, NEC, NVIDIA, NXP (antiga Philips), Samsung, Sharp, Texas Instruments e Yamaha* [13]

A tabela 2.1, abaixo, demonstra o grau de desenvolvimento atingido pela família Cortex-A8 (uma tabela mais completa que traz a evolução dos processadores ARM pode ser encontrada nos Anexo B).

²A mudança na sigla deveu-se a uma *joint venture* entre Acorn, Apple e VLSI Tec, que gerou uma empresa conhecida como *ARM Holdings* que tinha como objetivo investir e desenvolver a tecnologia da Acorn.

Tabela 2.1: ARM Cortex e Algumas de suas aplicações.

Família	Arquitetura	Núcleo	Aplicação
Cortex	ARMv7-A	Cortex-A5	-
		Cortex-A8	Texas Instruments OMAP3xxx series, SBM7000, Gumstix Overo Earth, Pandora, Apple iPhone 3GS , Apple iPod touch (3rd Generation), Apple iPad (Apple A4 processor), Archos 5, FreeScale i.MX51-SOC, BeagleBoard , Motorola Droid, Palm Pre, Rockchip RK2806 and RK2808, Samsung i8910, Book, Nokia N900, Meizu M9.
		Cortex-A9	-
		Cortex-A9 MPCore	Texas Instruments OMAP4430/4440, ST-Ericsson U8500, Nvidia Tegra2
	ARMv7-R	Cortex-R4(F)	Broadcom , TMS570 from Texas Instruments
	ARMv7-ME	Cortex-M4 ("Merlin")	-
	ARMv7-M	Cortex-M3	Texas Instruments Stellaris microcontroller family, Ember's EM3xx Series, Atmel AT91SAM3, Europe Technologies EasyBCU, Energy Micro's EFM32
ARMv6-M		Cortex-M0 ("Swift")	NXP Semiconductors NXP LPC1100, Triad Semiconductor, Melfas, Chungbuk Technopark, Nuvoton, austriamicrosystems
		Cortex-M1	Actel ProASIC3, ProASIC3L, IGLOO and Fusion PSC devices, Altera Cyclone III, other FPGA products are also supported

2.1.1.1 RS-232

No contexto de trabalho com a BB, o RS232 foi fundamental pois foi ele que permitiu a comunicação inicial com a placa (ainda sem sistema operacional) para que as configurações necessárias à inicialização correta da placa fossem efetuadas.

Amplamente utilizado em portas de computadores, o *Recommended Standard 232* é um padrão de comunicação serial³ que relaciona o DTE (*Data Terminal Equipment*) e o DCE (*Data Circuit-terminating Equipment*), quem envia e recebe dados. No total, o padrão recomenda um conector com 25 pinos, conhecido como *D-subminiature 25*, e especifica 20 diferentes sinais de comunicação; porém, observa-se uma tendência ao uso reduzido deste número de sinais, culminando em conectores menores (como o DB-9). Para se ter uma idéia da maleabilidade deste padrão, é possível definir uma conexão RS-232 mínima com apenas 3 pinos: Rx Tx e GND (Figura 2.3) e mais, caso a conexão seja *one way* (dados fluindo em sentido único) 2 pinos seriam suficientes. Apenas quando se deseja um controle do fluxo de dados, dois sinais adicionais precisam ser empregados: RTS e CTS(Figura 2.3) [10].

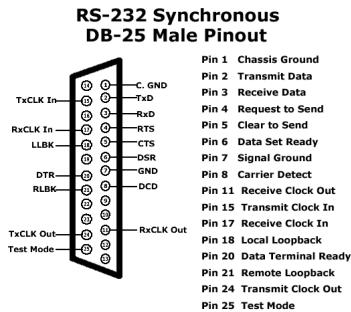


Figura 2.3: RS-232 conector DB-25 apresentando na totalidade os sinais de comunicação definidos pelo padrão em questão

2.1.1.2 Ethernet

Atualmente, conectividade é um recurso básico; tanto para compartilhar informação quanto para manter-se atualizado. Do ponto de vista dos sistemas linux (em especial a distribuição Angstrom, trabalhada aqui), ter acesso à rede significa obter mais facilmente os pacotes necessários ao perfeito funcionamento do sistema que será montado, e a tecnologia que rege este acesso é a Ethernet. O padrão *Ethernet*⁴ ganhou seu nome do antigo conceito grego de éter (do Latim, *aether* significa o mais puro ar) que seria a matéria que compunha o universo e demais coisas misteriosas⁵. Ethernet é uma família de tecnologia de redes baseadas em frame para LAN's (Local

³comunicação serial significa que é enviado um bit por vez, sequencialmente, através de um canal de comunicação (um barramento de computador, por exemplo).

⁴Na verdade, ethernet é um nome popular que permaneceu mesmo após a padronização da tecnologia pelo IEEE segundo o qual esta tecnologia seria o padrão 802.3[6].

⁵o ar que infla o corpo dos mortos era conhecido como Éter de Plutão

Area Network, inglês para “rede de área local”) e define um número de sinais e padrões de sinalização para a camada física do modelo OSI [2] (*Open System Interconnection*. Figura 2.4), bem como um formato comum de endereçamento e MAC⁶ na camada de “data link” .

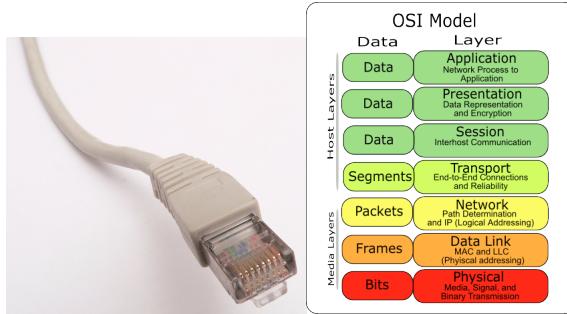


Figura 2.4: Cabo Ethernet RJ-45 e Modelo de camadas OSI

Ao longo dos anos (algo a partir dos anos 80), o padrão ethernet vem substituindo padrões concorrentes e firmando posição no meio de telecomunicações, sua versão para tecnologias ópticas já está em uso no oriente, sendo aplicada em rede ópticas passivas conhecidas como EPON’s e aguarda uma disputa com o padrão GM-PLS (*Generalized Multiprotocol Label Switching*⁷) que, inclusive, deve ser o padrão adotado pelo Brasil[4] nos próximos anos.

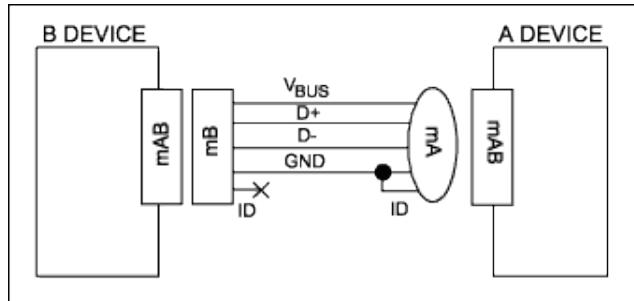
2.1.1.3 USB - *Universal Serial Bus*

Atualmente, os principais periféricos acoplados aos computadores utilizam o padrão USB como forma de comunicação, na BB não poderia ser diferente. Nesta subseção também é enfatizado o padrão OTG. É interessante ficar atento a esta parte pois ela é alvo de modificações durante os procedimentos práticos descritos no capítulo seguinte. O barramento serial universal começou a ser desenvolvido em 1994 por um grupo de sete empresas: *Compaq, DEC, IBM, Intel, Microsoft, NEC e Nortel*. A premissa básica do USB seria tornar a conexão de periféricos aos PC’s mais fácil, substituindo assim a vasta gama de conectores diferentes presentes até então. Além disso, o USB também implementaria uma classe para armazenamento em massa de informações, a *USB mass storage class* que encontra nos *pen-drives* seu maior exemplo.[15]

O USB 1.0 foi introduzido em 1996 e permitia uma taxa de transferência de dados de 12Mbps, no entanto, a versão USB adotada pela priemira vez em larga escala foi a 1.1 (1998). A epsecificação USB1.1 dividia os dispositivos em dois grandes grupo LS (de baixa velocidade: “*Low Speed*”) e FS (de alta velocidade: “*Full Speed*”) e alocava taxas de transferência diferentes para eles. Para os dispositivos FS, como discos rígidos, a taxa de transferência manteve-se igual à do USB1.0, 12Mbps; e para os dispositivos LS como joysticks e teclado, uma taxa menor foi designada, 1.5Mbps.[11]

⁶Media Access Control, endereço físico de um dispositivo

⁷é um padrão generalizado para telecomunicações bastante atual.



Esta figura ilustra como os papéis iniciais dos dispositivos são determinados pela orientação do cabo. A partir disso, eles podem trocar de papel sempre que necessário através do HNP (*Host-Negotiation-Protocol*).

Figura 2.5: USB-OTG

Em 2001, um esforço conjunto da *Hewlett-Packard*, *Intel*, *Lucent Technologies* (atual *Alcatel-Lucent*), *NEC* e *Philips* levou à padronização do USB2.0, agora com uma taxa de transferência de 480Mbps⁸. E mais recentemente, em 2010, os primeiros produtos que utilizam a tecnologia USB3.0 começaram a ser comercializados. Esta última tecnologia apresenta uma taxa significativamente maior, 5Gbps em modo “superspeed”, como resultado a tecnologia precisou alterar a forma física de seus conectores. Os cabos USB passarão a apresentar 2 fios para Vcc e GND, 2 fios para dados em modo normal, 4 fios para dados em modo “superspeed” e uma blindagem (que não era necessária nas especificações anteriores), porém os conectores serão retro-compatíveis. Vale destacar também que dispositivos corretamente configurados poderão drenar até 900mA da porta USB3.0, quase o dobro das anteriores, 500mA.[15]

Sobre os conectores USB, existem diversos formatos e tipos de conectores para este padrão mas, na especificação original havia apenas dois padrões:

- USB-A: este conector foi desenvolvido para ser ligado à uma porta de *downstream* do dispositivo principal (*host*). Neste tipo de conexão, dados fluem e energia é demandada do *host*. É comumente observado em cabos permanentemente fixados a um periférico, tal como teclado ou mouse.
- USB-B: neste tipo de conexão, embora também possa carregar ambos: dados e energia, é comum ver dispositivos utilizando a porta USB apenas como fonte de força.

Somado a tudo isso, uma expansão na especificação USB introduziu o conceito OTG (*on-the-go*) que implica na utilização da comunicação USB entre dois dispositivos que não são necessariamente *host's* (como pode ser visto na figura 2.5) de forma simples, o periférico que está com o cabo tipo A acoplado a si é dito dispositivo-A e passa a agir como principal (*host*) para o outro dispositivo, energizando a interface USB. Um dispositivo com cabo tipo B acoplado comporta-se como periférico, este é o comportamento padrão assumido por um dispositivo USB-OTG.[15]

⁸dispositivos que se utilizam desta taxa são ditos HS (“High Speed”)

2.1.1.4 *Cross-Compilation*

Escrever um programa seria uma tarefa consideravelmente mais complexa caso os compiladores não existissem, visto que as máquinas só entendem sua própria linguagem, a dita linguagem de máquina ou linguagem binária; o compilador exerce uma função crucial (de tradutor) neste sistema, sendo responsável por converter um texto escrito em uma determinada linguagem de programação para 0's e 1's. Agora, é comum que códigos sejam escritos para serem executados em máquinas de mesma arquitetura (ver 2.1) e mais comum ainda que sejam escritos para serem rodados na mesma máquina em que são escritos, isto é dito compilação nativa.^[5]

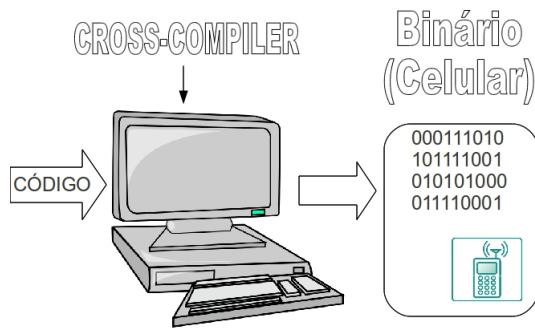


Figura 2.6: *Cross-Compiling*. Geração de binários através de um compilador instalado em uma máquina de arquitetura diferente da arquitetura alvo.

Quando se lida com PC's, a compilação nativa é completamente aceitável, é algo natural pois, onde mais se escreveria um programa? e, para que mais senão para rodar no computador? entretanto, quando o foco gira em torno de dispositivos não tão poderosos (em termos de velocidade de processamento) quanto o PC, fazer compilação nativa pode não ser tão razoável. Em celulares por exemplo, os smart-phones atuais são capazes até mesmo de rodar sistemas operacionais e apesar do contínuo aumento no clock de seus processadores (vem aumentando muito, o próprio OMAP35x está sendo utilizado em celulares), a grande maioria não está nem próxima da casa dos GHz; como as aplicações estão ficando cada vez mais complexas e até mesmo para a comodidade do programador e economia de tempo, é muito interessante deter a habilidade de gerar binários para alvos diferentes com a velocidade de um PC (como ilustrado na figura 2.6). Para a realização deste feito, é necessário possuir um tradutor para a arquitetura de interesse, este é o *Cross-Compiler*.

2.1.1.5 OE - *OpenEmbedded*

O Projeto *OpenEmbedded* (OE) é um *framework* de *software* criado com o objetivo principal de gerar distribuições Linux [9], mas não unicamente para sistemas embarcados. Foi criado originalmente por Chris Larson, Michael Lauer, e Holger Schurig, combinando os resultados obtidos pelo OpenZaurus com contribuições de

projetos como o Familiar Linux e OpenSIMPAD em uma base de códigos comum. O *OpenEmbedded* englobou estes projetos e é atualmente utilizado para gerar qualquer um deles através da mesma base de dados. Primariamente, o projeto mantém e desenvolve uma coleção de *BitBake recipes*, similar aos *ebuilds* do *Gentoo*. Os *bitbakes* consistem em uma fonte URL do pacote, dependências e opções de instalação e compilação. Durante o processo de construção, eles são utilizados para rastrear dependências, executar *cross-compile* (ver 2.1.1.4) e gerar novos pacotes de acordo com as configurações desejadas para uma arquitetura alvo. Também é possível criar imagens completas, consistindo de sistema de arquivos raiz e *kernel* (ver 2.2.1).

Durante o projeto, o OE foi utilizado diversas vezes, fosse para gerar uma imagem da distribuição angstrom [1] ou algum pacote especificamente.

2.2 Sobre Sistemas Operacionais e Outros Conceitos

É difícil encontrar uma definição isolada para sistema operacional (S.O.) além de ser um programa que é executado em modo kernel, e mesmo isso não é sempre verdade. Parte do problema é que os S.O.'s realizam basicamente duas funções distintas: prover àqueles que programam aplicações (e aplicações, naturalmente) um conjunto limpo e abstrato de recursos, ao invés de “confusos” ambientes de hardware e gerenciar estes mesmos recursos de hardware.[11]

Sendo um conceito complexo, optou-se por apresentar abaixo peças fundamentais de um sistema operacional para que a elucidação sobre o mesmo aconteça de maneira mais natural.

2.2.1 Kernel

O kernel é nada menos que o núcleo, o cerne de um sistema operacional. Trata-se de uma camada de baixo nível responsável por controlar e prover acesso⁹ à recursos essências de hardware como memória, dispositivos presentes e processador (como pode ser visto na figura 2.7)[11].

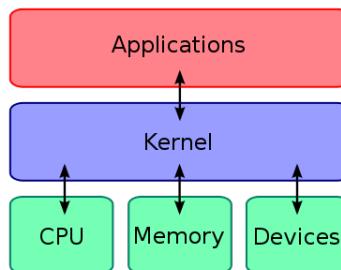


Figura 2.7: Kernel - camada de abstração mais baixa que provê acesso aos recursos de hardware mais essenciais para aplicações de mais alto nível.

⁹isto é geralmente feito através de mecanismos de comunicação inter-processos e System Calls

- Kernel monolítico: todos os serviços do Sistema Operacional (S.O.) rodam no contexto principal de execução e residem na mesma região de memória. Alguns programadores defendem o modelo como facilmente implementável, no entanto, se um acionador de dispositivo (*device driver*) estiver mal escrito pode derrubar o sistema inteiro.[11]
- MicroKernel: este estilo propõe implementar um kernel apenas com as funcionalidades mais básicas como: gerenciamento de memória, multi-tarefas e comunicação inter-processos. O restante das funcionalidades devem ser implementados no espaço do usuário, como resultado, este tipo de kernel é mais facilmente mantido que o monolítico. Porém, o grande número de System Calls e mudança de contexto podem deixar o sistema mais lento. O microKernel permite a implementação de grande parte do sistema como um programa qualquer, escrito em uma linguagem de alto nível; permite também que um outro S.O. seja utilizado sobre o mesmo kernel inalterado.[11]
- Kernel Híbrido: um meio-termo entre o kernel micro e monolítico propõe que alguns serviços sejam mantidos no contexto do kernel para aumentar a performance de execução.[11]
- NanoKernel: um kernel deste tipo delega ainda mais funções que o Microkernel, inclusive as mais básicas como gerenciamento de memória.
- ExoKernel: este tipo de kernel não abstrai o hardware em modelos teóricos. Ao invés disso, aloca recursos físicos (tais como tempo de processamento, páginas de memória e blocos de disco) para programas diferentes. Uma aplicação sendo executada sobre um exoKernel pode fazer referência a um sistema operacional biblioteca, este por sua vez simula as abstrações de S.O.'s bem conhecidos, ou permite o desenvolvimento de novas. Isto é bastante interessante em termos de otimização de código, visto que abstrações desnecessárias podem ser ignoradas.[11]

2.2.2 Sistemas de arquivos

Agora que o núcleo de um S.O. foi explicado, é impossível não mencionar sistemas de arquivos, afinal, quando um usuário é inquirido sobre o domínio que ele possui sobre um determinado sistema como *Windows* as respostas normalmente são relativas ao sistema de arquivos: copiar arquivos, renomeá-los, movê-los e uma série de outras operações comuns.

Toda aplicação de computador necessita armazenar e recuperar informações. enquanto um processo está rodando, ele pode armazenar uma quantidade limitada de informação dentro do seu espaço de endereçamento. Um segundo problema em manter informações dentro de um processo é que quando este processo termina, a informação se perde. Vale citar também que quando há muitos processos rodando eles frequentemente precisam acessar a informação (ou partes dela) ao mesmo tempo [11].

A maneira de resolver todos esses inconvenientes é tornar a informação independente em si, independente de qualquer processo.

Arquivos são um mecanismo de abstração que provêem formas de armazenar informações no disco rígido e depois recuperá-las. Mais formalmente, arquivos são unidades lógicas de informação criadas por processos, um disco pode conter milhares ou até mesmo milhões delas. Se cada arquivo fosse pensado como um tipo de espaço de endereçamento não se estaria muito longe da verdade, exceto que eles são usados para modelar o disco ao invés da memória principal (RAM). [11]

Os arquivos são gerenciados pelo sistema operacional. Como eles são estruturados, nomeados, acessados, usados, protegidos e implementados são detalhes importantíssimos a serem definidos quando um S.O. é implementado, e a parte do sistema operacional que lida com arquivos é conhecida como sistema de arquivos. [11]

Neste trabalho dois sistemas de arquivos são empregados do começo ao fim dos experimentos, são eles FAT e EXT3, que são melhor explicados abaixo:

- FAT - ou tabela de alocação de arquivos (*File Allocation Table*), neste modelo, uma tabela que contém os endereços de cada bloco que compõe o disco e informações sobre os arquivos é guardada na memória principal. O fato de ser uma tabela de tamanho fixo beneficia o acesso randômico a qualquer um dos blocos do disco, e estes blocos não precisam se preocupar em armazenar informações extra, mantendo seu tamanho original.[11]
- EXT3 - de forma bastante suscinta, o ext3 é um sistema de arquivos cuja estrutura base a qual um arquivo está associado chama-se *i-node*, que lista todas as características, atributos e endereços de disco dos blocos que compõem um arquivo. A principal vantagem dos i-nodes em relação à FAT é que o *i-node* precisa apenas estar na memória principal quando o arquivo ao qual ele corresponde está em uso. O EXT3 também implementa um “journaling file system”, uma espécie de diário atualizado continuamente de qualquer mudança que venha a ocorrer no sistema; assim, quando ocorre um falha, os dados podem ser recuperados tornando o sistema mais robusto.[11]

2.3 Cluster

Um *cluster* [11] é basicamente um conjunto de computadores interconectados (através de uma rede *ethernet*, por exemplo. Ver 2.1.1.2), que se “conhecem” e compartilham informações e tarefas (processamento de dados) obedecendo uma plataforma pré-definida (sistema distribuído). Existem diversas arquiteturas de *cluster*, numa das mais comuns, *Beowulf*, existe uma única máquina (mestra) que gerencia os “nós” escravos. Algumas categorias de cluster são descritas a seguir:

- *Compute Cluster* - Trata-se da idéia mais natural de *cluster*, esta associação de computadores é utilizada para processamento de tarefas e não operações orientadas a *I/O*, como serviços *web*. Aqui, os nós estão muito proximamente conectados e há uma grande quantidade de tarefas interdependentes, ou seja, a comunicação entre os nós é freqüente. Exemplos comuns de aplicação deste modelo são simulações de clima e simulações de batidas de veículos.



Figura 2.8: Cluster da Universidade de Boise, claramente um cluster *local*.

- *Grid Computing* - Os *grids* encontram-se no outro extremo em relação ao tipo de *cluster* anteriormente descrito, as máquinas pertencentes ao *grid*, geralmente, estão dispostas em locais geográficamente diferentes e não necessariamente pertencem à mesma arquitetura (ver 2.1), fazendo com que estes possam atingir dimensões elevadíssimas em termos de número de máquinas (*clusters* que apresentam máquinas de arquiteturas diferentes são conhecidos como heterogêneos). Portanto, é imprescindível que a tarefas sejam altamente coesas (contidas em si próprias) para que não haja uma condição de estrangulamento em algum ponto do arranjo. Exemplos de *grid* são: o projeto *Folding@home* que analiza dados utilizados por pesquisadores na busca da cura para doenças como câncer e Alzheimer; e o projeto *SETI@home*, este faz uso de mais de três milhões de máquinas, espalhadas pelo mundo, para analizar dados colhidos pelo radiotelescópio do observatório de Arecibo em busca de evidências de vida extraterrestre.[8]

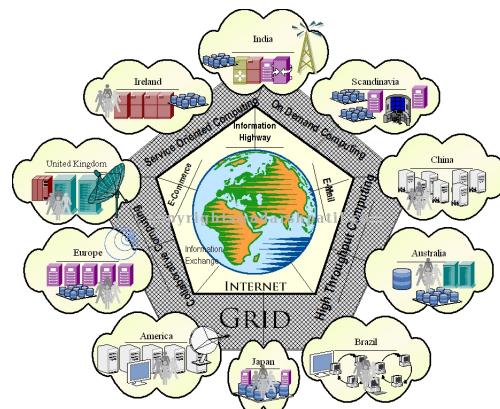


Figura 2.9: Exemplo de Grid com computadores espalhados ao redor do mundo trabalhando juntos.

Este é o ponto crucial do projeto proposto pelo aluno, associar máquinas como a BB para executar multiprocessamento de dados. Na subseção seguinte, a escolha da tecnologia que sustentará o cluster de *Beagle Boards*, MPICH, será justificada.

2.3.1 MPI e MPICH

O protocolo de passagem de mensagens, em inglês *message passing interface* (MPI), é um protocolo de comunicação bastante utilizado na programação de máquinas em ambiente paralelo; suas principais metas são, além da performance: escalabilidade e portabilidade[14]. A interface MPI deve prover uma topologia virtual essencial (sincronização e comunicação) entre um conjunto de processos que foram mapeados para as máquinas integrantes do *cluster*. O MPI também define *thread safe interfaces* que ajudam a evitar a manipulação errônea de algum estado dos processos dentro dos programas (a designação de qual processo será encaminhado a qual nó é tomada em tempo de execução).

MPICH foi a implementação MPI adotada por este trabalho e define-se pelo padrão MPI-1.1. Durante o processo de escolha, o MPICH2 (que se baseia no padrão MPI-2.0) também foi considerado, e descartado por ainda não suportar tradução de dados entre arquiteturas de hardware diferentes[14].

Capítulo 3

Métodos

As BB's foram obtidas por intermédio da Universidade de São Paulo que efetuou a importação das mesmas de uma empresa norte-americana chamada *Digikey*. Recém chegados, os pacotes continham apenas as placas BB, sem nenhum periférico para comunicação ou fonte para energizá-las. Um dos atrativos da BB é exatamente no que diz respeito à alimentação, a BB pode ser energizada via USB, no entanto, como a intenção era rodar a BB em modo *stand-alone* (sem depender de alguma outra máquina diretamente) e era de interesse reservar esta porta USB para comunicação com demais periféricos, um kit de alimentação externa foi criado.

A figura abaixo retrata uma vista superior da *Beagle Board* e será frequentemente referenciada durante este capítulo para ilustrar intervenções realizadas pelo aluno.

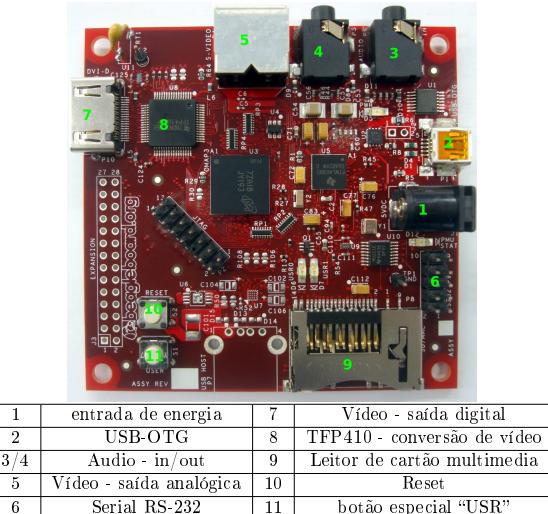


Figura 3.1: Regiões da Beagle Board trabalhadas durante este projeto

3.1 Kit de energia (Fonte de Alimentação)

Como já mencionado, é de interesse reservar a porta USB-OTG (ver 2.1.1.3) para comunicação com periféricos ao invés de alimentação. Na figura 3.1, a região de número “1” expõe um conector de energia que pode ser utilizado para este fim.

Para a montagem de um *kit*, como pode ser observado na figura 3.2, foram utilizados:

- 2 CI's LM7805 - reguladores de tensão para 5V
- 1 placa de *wire-up* (placa de cobre com furos)
- 1 cooler de alimentação 12V DC comum
- 1 dissipador de calor
- 1 fonte de tensão 12V DC, centro positivo, comum

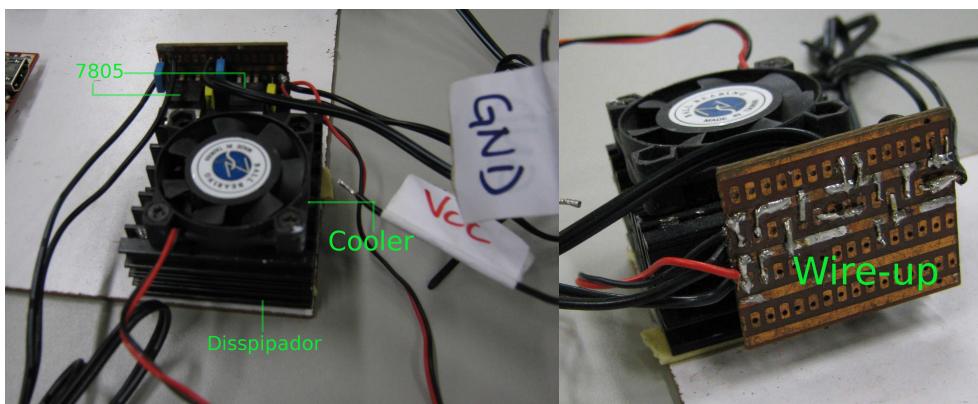


Figura 3.2: kit de energia e seus componentes

Primeiramente, um pedaço de placa *wire-up*, suficiente para acomodar os reguladores de tensão foi cortado e devidamente trilhado com solda (para a montagem do circuito regulador de tensão, utilizou-se o modelo proposto pelo datasheet do LM7805 que pode ser verificado no Anexo G). Em seguida, a partir da fonte de tensão, foram postos em paralelo, o *cooler*, um 7805 e depois o outro (como no esquema da figura 3.3).

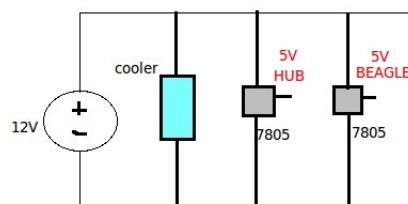


Figura 3.3: Esquema de montagem do kit de energia ilustrando como devem estar dispostos os componentes

Finalmente, os reguladores de tensão foram parafusados ao dissipador de calor para evitar qualquer anomalia de corrente que venha a causar seu mau funcionamento. Segundo as especificação do LM7805 [7], os circuito integrado deve funcionar bem com até 400mA de corrente, mais que isso ele esquentará demais. Isto completa a montagem do kit para fornecimento de energia da BB.

3.2 Distribuição LINUX Utilizada

O próximo passo na montagem da plataforma de trabalho foi designar uma distribuição linux compatível com a arquitetura do OMAP3530. Felizmente, o Angstrom linux já estava preparado para dar suporte à BB, e de seu website [1] foi possível obter arquivos extremamente úteis para um primeiro teste como:

- um kernel linux já compilado
- o *u-boot*, ou *bootloader universal* (uma ferramenta que provê o básico para a interação primária com diversos dispositivos, neste caso, a BB)
- uma imagem do sistema operacional (inclusive com *X-windows* instalado)
- *x-loader* (é executado durante o boot da BB, normalmente já está guardado na memória NAND da BB, mas em alguns casos é necessário atualizá-lo)

A partir de então, um cartão SD (figura 3.4) foi adquirido pelo laboratório e teve sua geometria alterada para imitar a de um disco rígido comum, ou seja, 255 cabeças e 63 setores de 512Mb cada. No Anexo F, é possível verificar como a ferramenta *fdisk* foi utilizada para realizar esta alteração. É importante lembrar que, todos estes procedimentos relativos ao cartão SD estão sendo realizados no computador do laboratório através de um leitor de cartão presente na máquina. Por último, o cartão de memória foi particionado em dois: uma partição FAT (ver 2.2.2) menor que auxilia no *boot* do sistema e, uma partição EXT3 (ver 2.2.2) maior que acomoda o sistema de arquivos em si.



Figura 3.4: Cartão de memória onde o sistema de arquivos será acomodado

O próximo passo foi copiar para dentro da partição FAT um arquivo binário chamado MLO que auxiliará no processo de *boot* (este processo será explicado melhor em 3.3), é importante que este binário seja a primeira coisa a ser movida para dentro cartão, ocupando assim o primeiro setor dele; copiou-se também para esta partição o Kernel Linux e um outro arquivo binário *u-boot* (também requerido pelo processo de *boot*). Já na partição EXT3, a imagem do sistema de arquivos foi extraída. Isto conclui todas as operações relativas ao cartão multimedia.

3.3 Plataforma de Trabalho

Até aqui, já estava a disposição do aluno uma kit de energia, e um cartão SD devidamente alterado e preparado com a distribuição Angstrom Linux. Restava configurar as variáveis de ambiente corretas na BB para que a inicialização do sistema operacional fosse realizada a partir do cartão SD e que a saída de vídeo digital fosse utilizada, bem como acoplar os periféricos à porta USB da BB. Os textos seguintes cuidam destes últimos entraves.

3.3.1 Confecção dos Cabos Seriais

O primeiro contato com a placa BB foi realizado através de comunicação serial RS-232, como indicado na figura 3.1 pela região “6”. Foram adquiridos então, dois conectores de 9 pinos e um cabo *flat* (10 fios) que foram utilizados para montar o cabo mostrado na figura 3.5 (o padrão do conector DB-9 foi seguido e também pode ser observado nesta figura).

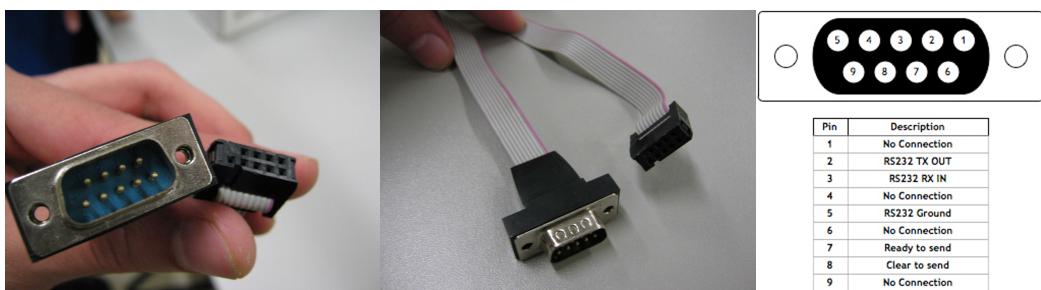


Figura 3.5: Cabo serial montado pelo aluno para primeira interação com a BB

É válido notar que apenas 9 pinos são utilizados na conexão serial, portanto o pino 10 do conector serial da BB deve ser ignorado (pode ser dobrado ou mesmo cortado).

Também foi necessário fazer uma pequena adaptação do conector USB no hub adquirido pelo laboratório para o padrão miniUSB. A intervenção foi bem simples¹. Em um cabo USB qualquer (excluindo o padrão 3.0)

¹apesar de ter levado um tempo considerável dada a falta de experiência do aluno com soldagem.

existem 4 fios, 2 para alimentação da interface e 2 para troca efetiva de dados; ao observar a figura 2.5, vê-se a adição de mais um fio, o “id” que encontra-se interrompido do lado do periférico e portanto, idêntico a um USB padrão. Efetuou-se então a troca do conector USB padrão do hub por um conector miniUSB tipo B como se pode ver na figura 3.6 (todas as soldagens foram isoladas com termo-retrátil ou reforçadas com cola de silicone quando conveniente).

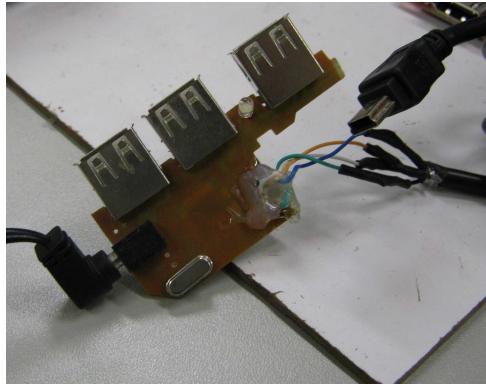


Figura 3.6: HUB USB com conector trocado para o padrão miniUSB

Por último, para reservar definitivamente à USB-OTG sua função como *host*, uma gota de solda foi utilizada para curto-circuitar as trilhas 4 e 5 deste conector indicado na figura 3.1 pela região “2”, curto-circuitando assim os sinais correspondentes a “id” “gnd” na figura 2.5 e definindo portanto, um dispositivo tipo “A”.

3.3.2 Processo de *BOOT*

Chegou a hora de finalmente fazer um primeiro contato com a BB, mas primeiramente é importante entender a função de cada um dos objetos presentes no cartão de memória:

- *x-loader* - é dito um *bootloader* de primeiro estágio e sua função é preparar o sistema para rodar o estágio seguinte e transferir o controle para ele, que seria o *u-boot*. O *x-loader* é compacto e cabe na memória interna ROM do OMAP, ele ativa o controle da memória externa para poder carregar o *u-boot*, que demanda mais espaço. O *x-loader* é normalmente transferido via serial porém, na BB, um arquivo diferente é utilizado: o MLO.
- MLO - quando não há resposta aos pulsos emitidos pelo OMAP na porta serial, ele busca automaticamente no cartão de memória por um arquivo especial, o MLO, que é na verdade um *x-loader* com um cabeçalho indicando uma posição de memória onde este deve ser carregado e também seu tamanho).
- *u-boot* - é um *bootloader* de segundo estágio que tem como função carregar o kernel também presente na partição FAT do cartão SD para finalmente tornar disponível todos os recursos de hardware, inclusive a

memória RAM e possibilitar o *boot* do linux. O *u-boot* é transferido para a memória principal pelo MLO e em seguida tem o controle passado para si.

sabendo disto, o cabo serial montado em 3.3.1 é utilizado, em conjunto com um conector DB9FxDB9F para conectar a BB ao computador do laboratório (figura 3.7).



Figura 3.7: *Beagle Board* conectada ao computador do laboratório via o cabo serial montado pelo aluno.

Bastou configurar o software *minicom* para uma comunicação de serial 115200 baud/s, 8 bits, sem paridade e 1 stop-bit (maiores detalhes podem ser encontrados no Anexo E) e o console serial da BB, rodando o *u-boot* fez-se disponível (figura 3.8).

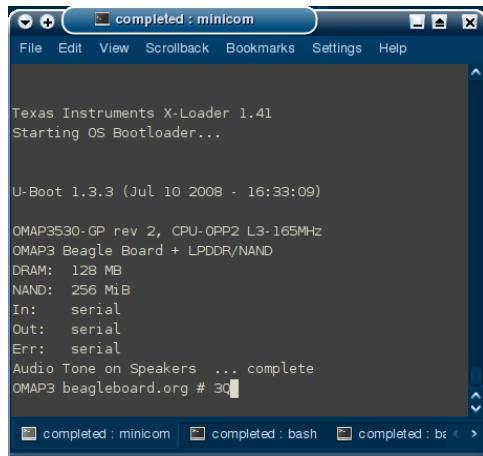


Figura 3.8: Imagem do software minicom exibindo o console serial da BB executando o u-boot

Trocou-se então as varáveis de ambiente de interesse “bootargs” e “bootcmd” que correspondem aos argumentos e ao comando de *boot* respectivamente (nos Anexo E, as linhas de código necessárias a isto estão explicitadas).



Figura 3.9: *Beagle Board* rodando o gerenciador gráfico enlightenment

Finalmente todo o ajuste básico estava completo, após o reinicio do sistema, o boot passou a ser efetuado pelo cartão SD, eliminando a necessidade da conexão serial com o computador. Na figura 3.9, pode-se ver a BB rodando o Angstrom Linux com *Enlightenment* como gerenciador gráfico.

3.4 Ajuste Fino

Em teoria, a partir deste ponto, faltaria apenas repetir o mesmo procedimento para a outra placa BB. No entanto, deve-se lembrar que todo o material utilizado, em termos de software, vieram de um exemplo fornecido pelo time de desenvolvimento do Angstrom Linux e portanto, possuia muito mais pacotes que os realmente necessários, o kernel não estava tão atualizado quanto poderia e assim alguns módulos poderiam estar também desatualizados. Para corrigir estes detalhes, uma “peregrinação” de nível muito mais baixo foi traçada, envolvendo desde a obtenção do código fonte do kernel até a compilação assistida do driver para o adaptador USB/Ethernet que foi utilizado para prover conectividade à BB.

3.4.1 Utilizando o OpenEmbedded (OE)

Após navegar em diversas listas de desenvolvimento, esta ferramenta, OE, chamou bastante atenção, como explicado em 2.1.1.5, o OE se utiliza de arquivos chamado *recipes* e um *cross-compiler* (ver 2.1.1.4) para criar distribuições linux inteiras ou pacotes separados. Em diversas situações foi feito uso do OE, o caso de maior sucesso foi para gerar distribuições linux sem servidor X, ou seja, *console-only*; outra tentativa diz respeito a escrever uma *recipe* para o MPICH que não está incluída no trabalho dado seu fracasso.

O OpenEmbedded foi obtido através de um repositório GIT (ferramenta de versionamento) clonado para o computador do laboratório, foi então instalada sua principal ferramenta: *bitbake*, que é responsável pela interpretação e execução das *recipes* (maiores detalhes sobre a instalação e uso do OE podem ser encontradas no Anexo D deste trabalho); finalmente, um perfil (ver Anexo D) descrevendo a arquitetura alvo para a qual todo pacote deveria ser compilado foi definido e carregado sempre que se desejo utilizar o OE.

O OE foi ser utilizado também para construir as ferramentas de *boot* como o *u-boot* a partir de fontes mais atualizados.

3.4.2 Compilando o Kernel e Drivers Separadamente

Uma das grandes dificuldades encontradas no trabalho foi como dar suporte ao adaptador USB/Ethernet (figura 3.10), por uma infelicidade, o kernel em sua versão 2.6.26 (que foi herdado dos arquivos de exemplo do time de desenvolvimento do Angstrom Linux) não suportava o *chipset* do adaptador que correspondia ao módulo de nome dm9601. Durante este projeto, diferentes soluções para este problema foram encontradas porém, nenhuma definitiva.

A primeira correção tentada foi obter o código fonte do kernel linux, já modificado para o OMAP3 (este também encontra-se em um repositório GIT e como qualquer processo que envolva linhas de código, está descrito nos Anexo D deste trabalho); obter o código fonte do módulo dm9601 e efetuar algumas modificações em seu interior para dar suporte ao adaptador em questão. Feito isso, incluiu-se o fonte alterado na respectiva pasta de módulos do kernel OMAP e através da linha de comando, foi ordenado que apenas os módulos fossem reconstruídos de maneira independente (ou seja, não acoplados ao kernel linux). Assim, ao final do processo, um arquivo binário referente ao módulo em questão estava disponível na pasta do kernel e foi prontamente copiado e incluido no cartão de memória onde pôde ser carregado através do comando *modprobe*.



Figura 3.10: Adaptador USB/Ethernet com *chipset* correspondente ao módulo linux dm9601

Apesar do OE já apresentar um *cross-compiler*, um outro chamado *CodeSourcery* (que possui otimizações para o Cortex-A8) foi instalado devido a dificuldade de se trabalhar com o *cross-compiler* do OE fora da sua árvore de diretórios.

Algum tempo depois, a partir do kernel 2.6.27, o módulo correspondente ao adaptador presente no laboratório passou a ter suporte do kernel, e as imagens podiam ser geradas no OE sem o contratempo de precisar recompilar este módulo separadamente.

Como última alternativa, o time do angstrom (sempre muito a frente no que diz respeito a BB) criou uma “casca” online dentro de seu site, onde se podia utilizar o OE de uma forma bem simples e intuitiva, este é o projeto *Narcissus*: ao fim de uma configuração feita via *checkbox’s*, o *narcissus* disponibilizava para download tanto a imagem linux personalizada e a *recipe* utilizada para gerá-la, quanto uma imagem de cartão SD pronta para ser grava em um dispositivo deste tipo.

Em todos os casos, o driver não funcionou bem e apresentou problemas relacionados ao tamanho dos pacotes recebidos e, mesmo tendo seu MTU (configuração do tamanho dos pacotes de dados recebidos através da interface de rede) alterado para um padrão ethernet (1492 ou 1500) o mesmo continuou a quebrar (em inglês utiliza-se o termo *crash*) após um uso prolongado da conexão.

Daí a vantagem de se ter o módulo construído separado do kernel, quando ocorreram *crash’s*, bastou remover o módulo com o comando *rmmmod* e carregá-lo novamente; agora, nas situações em que o módulo foi construído *built-in* (junto com o kernel), um *crash* no módulo significou por várias vezes um *crash* no kernel. E isto tem um nome bastante temido entre os que trabalham com linux, chama-se *kernel panic* e a única forma de resolvê-lo é recarregando o kernel (ou seja, reiniciando a máquina).

A única razão para insistir no uso do mesmo adaptador USB ethernet é que, no Brasil, todos os chipsets dos adaptadores vendidos (encontrados pelo aluno) eram os mesmos e por consequência, utilizavam o mesmo módulo.

3.5 Escolha da Implementação MPI e Montagem do Cluster

Foram testadas diversas implementações MPI, destaca-se o OpenMPI por já ser familiar a alguns alunos do laboratório, o que atenuaria bastante a curva de aprendizado da ferramenta. Infelizmente, após diversas tentativas, não foi possível traduzir o OpenMPI para a arquitetura ARMv7-A nem através de *cross-compiling*, nem através de compilação nativa (ver 2.1.1.4).

Também foi avaliado o MPICH2 que se baseia no padrão MPI 2.0 (ver 2.3.1) que também foi descartado pela impossibilidade de adaptação à arquitetura alvo.

Finalmente, o MPICH 1.2.27 foi passível de tradução. A compilação nativa é mais facilmente realizável pois o processo de instalação da ferramenta² executa testes, e quando se tenta executar binários em de outra arquitetura a instalação obviamente falha pois os testes falham; logo para se executar um compilação cruzada,

²padrão linux: primeiro executa-se um script de configuração, que checa se todas as dependências necessárias ao pacote estão disponíveis e cria *Makefiles*; depois o comando *make* executa os *Makefiles* para gerar binários; por fim, o comando *make install* copia os binários e demais arquivos para a árvore linux em questão

é necessário alterar os scripts de configuração do MPICH para que ignorem estes testes (scripts com algo em torno de 16000 linhas). Os dois métodos foram provados viáveis.

O MPICH é interessante do ponto de vista de *cross-compiling*. Enquanto o OpenMPI utiliza um binário chamada *mpicc* para compilar códigos que utilizem o padrão MPI, o *mpicc* do MPICH é apenas um script que indica como o que um compilador comum deve fazer para traduzir um código MPI para binário. De posse desta informação, bastou indicar neste script que o *cross-compiler* deve ser usado para se obter binários relativos a uma outra arquitetura.

Por fim, as plataformas de trabalho *Beagle Board*, tiveram o adaptador USB/Ethernet conectados a si através do hub USB (pela porta USB-OTG) e em seguidas foram conectadas à uma rede comum. Depois foi utilizado um hub ethernet de 100 Mbps para conectar as plataformas até que se pudesse obter um hub ethernet de qualidade com taxas de comunicação Gigabit (visto na figura 3.11).



Figura 3.11: HUB Ethernet com elevada taxa de comunicação. Utilizado para amenizar a latência da rede.

Também foi obtido um porto do MPICH para a arquitetura ARMv5-TE, referente a um *kit* de desenvolvimento DSP (figura 3.12) presente no laboratório.



Figura 3.12: *Kit* DSP L137 para o qual o MPICH foi portado também.

3.6 Resultados

Foram obtidos desde o início do trabalho:

- Duas plataforma de trabalho compostas (cada uma) por: uma BB com porta USB-OTG devidamente

modificada para exercer o papel de *host*; um *kit* de energia composto por *cooler*, dissipador de calor e dois reguladores de tensão de 5V e uma fonte de 12V DC; um hub USB com alimentação externa (vinda do *kit* de energia). As BB's foram configuradas para que a saída de vídeo digital fosse utilizada³ (mais detalhes no Anexo E), portanto, podem ser ligadas a qualquer monitor compatível. Na figura 3.13, pode-se ver a forma final da plataforma de trabalho BB.

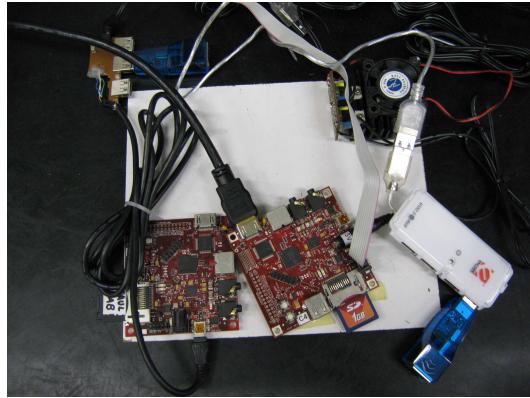


Figura 3.13: Plataforma de trabalho em sua forma final

- Versões pré-compiladas do MPICH para x86, ARMv7-A e ARMv5TE.
- Dois cartões de memória SD com distribuição Angstrom linux customizada.
- As habilidades necessárias para associar as plataformas de trabalho para que funcionem em paralelo, graças ao MPICH.
- Foram compilados e executados alguns testes bem simples apenas para esclarecer qualquer dúvida de que o MPICH havia sido portado corretamente para a BB. Um dos candidatos escolhidos para este propósito foi uma implementação paralela de cálculo do “Pi” (código fonte disponível no Anexo C) que foi corretamente executada.

3.6.1 Cluster Heterogêneo

Em certo ponto do trabalho, já tendo portado o MPICH para a arquitetura ARMv7-A, decidiu-se checar também a possibilidade de traduzi-lo para a arquitetura ARMv5-TE (ver 3.5) isto porque, no laboratório, havia dois *kits* DSP cujas arquiteturas correspondiam a esta. A figura 3.14, retrata uma configuração atingida com os *kits* DSP, as duas plataformas BB e o computador do laboratório.

³na figura 3.1, a região “8” indica um componente bastante interessante, o TFP410. Este componente é capaz de receber os dados em formato RGB e convertê-los para o padrão de vídeo HDMI, presente nos monitores de alta definição.

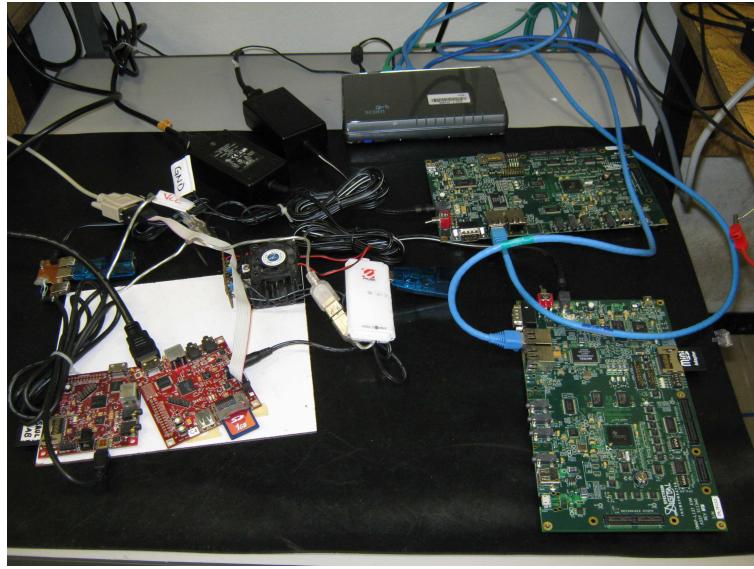


Figura 3.14: Cluster Heterogêneo formado pelo computador do laboratório, duas BB's e dois *kits DSP*

Apesar da baixa capacidade de processamento de dados (quando o uso do DSP não é feito), a implementação desta associação foi um sucesso e uma aplicação sendo executada por ela pode ser vista na figura 3.15.

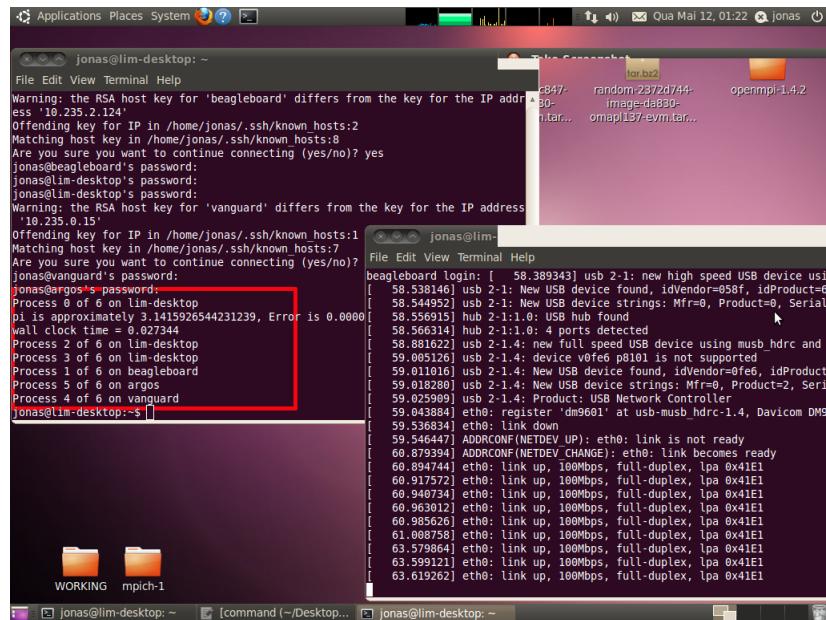


Figura 3.15: Imagem do MPI executando o programa de cálculo paralelo de PI

Capítulo 4

Conclusões

O projeto foi em todos os aspectos, bastante inovador, e apresentou grau de dificuldade médio para o aluno, visto que o mesmo não possuia conhecimentos anteriores nem de linux, nem de sistemas embarcados. Além disso, devido ao OMAP3530 ser um processador recente, as informações sobre este (na internet sobretudo) são escassas e, em sua maioria, pouco organizadas.

A confecção do cabo DB9 (ver figura 3.5), adaptação do conector do hub USB de padrão para mini e a compilação cruzada de um driver (realizada manualmente) também merecem destaque, pois fizeram com que um aprofundamento significativo em linux embarcado fosse atingido. O contato com a ferramente OE foi um dos pontos fortes do projeto, pois é largamente empregada na área de tecnologia embarcada e tem potencial para poupar o desenvolvedor de muito trabalho. Por exemplo, através do *bitbake* o OE baixa automaticamente o código fonte do kernel desejado, aplica todos os patches necessários a uma arquitetura alvo e ainda executa toda a parte de *cross compile*.

É indispensável citar também a revisão conceitos abordados pelo projeto, tais como: arquitetura de computadores, tecnologias de comunicação (serial, usb) e sistema operacional.

Como dito em 3.5, em um primeiro caso, quando as placas estavam conectadas à rede do Departamento de Engenharia Elétrica, a latência influiu muito nos resultados. Mais tarde, com o hub de 100Mbps, esta latência diminuiu, porém, ainda era mais rápido executar o programa localmente em apenas um dos nós. Só com o hub GigaBit é que foi possível se ter idéia da eficiência do *cluster*. Devido aos problemas encontrados com o chipset do adaptador USB/Ethernet (ver 3.4.2) não foi possível obter medidas para caracterizar de forma satisfatória o desempenho do *cluster*.

Em um determinado ponto, um cluster heterogêneo (ver 2.3) composto por, além da BB, mais dois kits de DSP de arquitetura ARMv5TE e o computador do laboratório (x86) esteve disponível. Apesar do porte do MPICH ter sido um sucesso, a baixa capacidade de processamento dos kits quando não fazem uso do DSP mostrou-se um inconveniente (300 MHz), no entanto, isto prova o potencial deste projeto.

Anexo A

Tabela 4.1: Especificações da Beagle Board e do OMAP3530

Beagle Board	
Processador (OMAP3530)	<ul style="list-style-type: none"> • 720MHz Cortex-A8 • 430MHz DSP • On-Chip L1/SRAM - 112 KB (DSP),32 KB (ARM Cortex-A8) • On-Chip L2/SRAM - 96 KB (DSP),256 KB (ARM Cortex-A8) • RAM 64 KB • ROM - 16 KB (DSP),32 KB (ARM Cortex-A8) • EMIF - 1 32-Bit SDRC,1 16-Bit GPMC • External Memory Type Supported - LPDDR,NOR Flash,NAND flash,OneNAND,Asynch SRAM • DMA - 64-Ch EDMA,32-Bit Channel SDMA • Video Port (Configurable) - 1 saída dedicada, 1 entrada dedicada • Acelerador Gráfico - Sim • MMC/SD - 3 • McBSP - 5 • Pinos/encapsulamento - 423FCBGA, 515POP-FCBGA • POP Interface - Sim • I2C - 3 • McSPI - 4 • HDQ/1-Wire - 1 • UART - 3 • USB - 2 • Timers - 12 32-Bit GP,2 32-Bit WD • Core Supply (Volts) - 0.8 V to 1.35 V • IO Supply (Volts) - 1.8 V,3.0 V (MMC1 apenas) • Operating Temperature Range (°C) - 0 to 90,-40 to 105
Memória RAM	128MB
Memória NAND	256MB
Consumo de Pico	2W

Anexo B

Tabela 4.2: Evolução dos processadores ARM

Família	Arquitetura	Núcleo	Aplicação
ARM1	ARM1 (obsoleto)	ARM1	segundo processador da BBC Micro
ARM2	ARMv2 (obsoleto)	ARM2	Acorn Archimedes, Chessmachine
	ARMv2a (obsoleto)	ARM250	Acorn Archimedes
ARM3	ARMv2a (obsoleto)	ARM2a	Acorn Archimedes
ARM6	ARMv3 (obsoleto)	ARM60	3DO Interactive Multiplayer, Zarlink GPS Receiver
		ARM600	-
		ARM610	Acorn Risc PC 600, Apple Newton 100 series
ARM7	ARMv3 (obsoleto)	ARM700	Acorn Risc PC prototype CPU card
		ARM710	Acorn Risc PC 700
		ARM710a	Acorn Risc PC 700, Apple eMate 300
		ARM7100	Psion Series 5
		ARM7500	Acorn A7000
		ARM7500FE	Acorn A7000+ Network Computer

ARM7TDMI	ARMv4T	ARM7TDMI(-S)	Game Boy Advance, Nintendo DS, Apple iPod, Lego NXT, Atmel AT91SAM7, Juice Box, NXP Semiconductors LPC2000 and LH754xx
		ARM710T	Psion Series 5mx, Psion Revo/Revo Plus/Diamond Mako
		ARM720T	Psion Series 5mx, Psion Revo/Revo Plus/Diamond Mako
		ARM740T	-
	ARMv5TEJ	ARM7EJ-S	-
StrongARM	ARMv4	SA-110	Apple Newton 2x00 series, Acorn Risc PC, Rebel/Corel Netwinder, Chalice CATS
		SA-1100	Psion netBook
		SA-1110	LART (computer), Intel Assabet, Ipaq H36x0, Balloon2, Zaurus SL-5x00, HP Jornada 7xx, Jornada 560 series, Palm Zire 31
ARM8	ARMv4	ARM810	Acorn Risc PC prototype CPU card
ARM9TDMI	ARMv4T	ARM9TDMI	-
		ARM920T	Armadillo, Atmel AT91SAM9, GP32, GP2X (first core), Tapwave Zodiac (Motorola i. MX1), Hewlet Packard HP-49/50 Calculators, Sun SPOT, Cirrus Logic EP9302, EP9307, EP9312, EP9315, Samsung S3C2442 (HTC TyTN, FIC Neo FreeRunner), Samsung S3C2410 (TomTom navigation devices)
	ARMv4T	ARM922T	NXP Semiconductors LH7A40x

ARM9TDMI	ARMv4T	ARM940T	GP2X (second core), Meizu M6 Mini Player
ARM9E	ARMv5TE	ARM946E-S	Nintendo DS, Nokia N-Gage, Canon PowerShot A470, Canon EOS 5D Mark II, Conexant 802.11 chips, Samsung S5L2010
		ARM966E-S	ST Micro STR91xF, includes Ethernet
		ARM968E-S	NXP Semiconductors LPC2900
ARM9E	ARMv5TEJ	ARM926EJ-S	Mobile phones: Sony Ericsson (K, W series); Siemens and Benq (x65 series and newer); LG Arena; OMAP-L137 , OMAP-L138; Freescale i.MX21, i.MX27, Atmel AT91SAM9, NXP Semiconductors; Buffalo TeraStation Live (NAS); Telechips TCC7801, TCC7901; ZiiLABS' ZMS-05 system on a chip; Western Digital MyBook I World Edition.
	ARMv5TE	ARM996HS	-
ARM10E	ARMv5TE	ARM1020E	-
		ARM1022E	-
	ARMv5TEJ	ARM1026EJ-S	Western Digital MyBook II World Edition; Conexant so4610 and so4615 ADSL SoC
XScale		80200/IOP310/ IOP315	-
		80219	Thecus N2100
		IOP321	Iyonix
		IOP33x	-
		IOP34x	-
		PXA210/PXA250	Zaurus SL-5600, iPAQ H3900, Sony CLIÉ NX60, NX70V, NZ90

XScale	ARMv5TE	PXA255	Gumstix basix & connex, Palm Tungsten E2, Zaurus SL-C860, Mentor Ranger & Stryder, iRex ILiad
		PXA263	Sony CLIÉ NX73V, NX80V
		PXA26x	Palm Tungsten T3
		PXA27x	Gumstix verdex, Trizeps-Modules, PXA270 COM, HTC Universal, HP hx4700, Zaurus SL-C1000, 3000, 3100, Rokr E6, Fujitsu Siemens LOOX N560, Toshiba Portégé G500, Treo 650-755p, Zipit Z2, HP iPaq 614c Business Navigator.
		PXA800(E)F	-
		PXA3XX ("Monahans")	Samsung Omnia
		PXA900	Blackberry 8700, Blackberry Pearl (8100)
		IXC1100	-
		IXP2400/IXP2800	-
		IXP2850	-
ARM11	ARMv6	ARM1136J(F)-S	Texas Instruments OMAP2420 (Nokia E90, Nokia N93, Nokia N95, Nokia N82)
	ARMv6T2	ARM1156T2(F)-S	-
	ARMv6KZ	ARM1176JZ(F)-S	Apple iPhone (original and 3G), Apple iPod touch (1st and 2nd Generation), Conexant CX2427X, Motorola RIZR Z8, NVIDIA GoForce 6100; Telechips TCC9101, TCC9201
	ARMv6K	ARM11 MPCore	Nvidia APX 2500

Cortex	ARMv7-A	Cortex-A5	-
		Cortex-A8	Texas Instruments OMAP3xxx series, SBM7000, Gumstix Overo Earth, Pandora, Apple iPhone 3GS , Apple iPod touch (3rd Generation), Apple iPad (Apple A4 processor), Archos 5, FreeScale i.MX51-SOC, BeagleBoard , Motorola Droid, Palm Pre, Rockchip RK2806 and RK2808, Samsung i8910, Book, Nokia N900, Meizu M9.
		Cortex-A9	-
	ARMv7-M	Cortex-A9 MPCore	Texas Instruments OMAP4430/4440, ST-Ericsson U8500, Nvidia Tegra2
		Cortex-R4(F)	Broadcom , TMS570 from Texas Instruments
		Cortex-M4 ("Merlin")	-
		Cortex-M3	Texas Instruments Stellaris microcontroller family, Ember's EM3xx Series, Atmel AT91SAM3, Europe Technologies EasyBCU, Energy Micro's EFM32
		Cortex-M0 ("Swift")	NXP Semiconductors NXP LPC1100, Triad Semiconductor, Melfas, Chungbuk Technopark, Nuvoton, austriamicrosystems
		Cortex-M1	Actel ProASIC3, ProASIC3L, IGLOO and Fusion PSC devices, Altera Cyclone III, other FPGA products are also supported

Anexo C

```
[CPI.C]
#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f(double);

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

int main(int argc,char *argv[])
{
    int done = 0, 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 on %s\n",
            myid, numprocs, processor_name);

    n = 0;
    while (!done)
    {
        if (myid == 0)
        {
/*
            printf("Enter the number of intervals: (%s quits) ");
            scanf("%d",&n);
*/
        if (n==0) n=10000; else n=0;

        startwtime = MPI_Wtime();
    }
    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
    if (n == 0)
        done = 1;
    else
    {
        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)
{
    printf("pi is approximately %.16f, Error is %.16f\n",
           pi, fabs(pi - PI25DT));
    endwtime = MPI_Wtime();
    printf("wall clock time = %f\n", endwtime-startwtime);
    fflush( stdout );
}
MPI_Finalize();
return 0;
}
```

Figura 4.1: Código fonte utilizado nos testes dos *cluster's*

Anexo D

- **Kernel adaptado para OMAP:**

Comandos para efetuar a clonagem do repositório

```
$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/tmlin/lina-omap-2.6.git
```

```
$ git checkout 58cf2f1 \\ para ir ao ramo correspondente ao kernel 2.6.29 (outros podem ser selecionados, inclusive mais recentes)
```

```
$ make ARCH=arm CROSS-COMPILE=arm-none-gnueabi- dist clean \\ limpa qualquer configuração prévia
```

```
$ make CROSS_COMPILE=arm-none-linux-gnueabi- omap3_beagle_defconfig \\ configura o kernel para se adaptar à BB
```

```
$ make CROSS_COMPILE=arm-none-linux-gnueabi- menuconfig \\ pode ser usado para alterar as configurações padrão (inclusive como se deseja construir os módulos (built-in ou não))
```

```
$ make CROSS_COMPILE=arm-none-linux-gnueabi- uImage modules \\ gera o kernel na pasta arch/arm/boot/ directory e também constrói os módulos
```

```
$ make ARCH=arm CROSS-COMPILE=arm-none-gnueabi- INSTALL_MOD_PATH=/segunda/partição/do/cartão_SD modules_install” \\ instala os módulos no cartão SD
```

- **OpenEmbedded**

Procedimentos para instalação:

```
$ apt-get install ccache sed wget cvs subversion git-core coreutils unzip texi2html texinfo libsdl1.2-dev docbook-utils gawk help2man diffstat gtk-doc-tools file g++ python-psycod minicom build-essential libncurses5-dev python-dev python-pysqld2 quilt mkimage”
```

```
$ git clone git://git.openembedded.net/openembedded
```

```
$ cd openembedded
```

```
$ git checkout origin/stable/2009 -b stable/2009
```

```
$ git pull \\ atualiza o repositório git
```

O arquivo de perfil da BB: home/oe/beagleboard/beagleboard/Profile.sh:

```
export OE_HOME=$HOME/oe
export MY_OE_CONF="beagleboard"
export BBPATH=$OE_HOME/beagleboard/:$OE_HOME/beagleboard/$MY_OE_CONF:$OE_HOME/openembedded
export BBFILES="$OE_HOME/openembedded.org/openembedded/recipes/*/*.bb"
export BB_ENV_EXTRAWHITE="MACHINE DISTRO ANGSTROM_MODE ANGSTROMLIBC OE_HOME"
export PATH=$OE_HOME/openembedded.org/openembedded/bitbake/bin:$PATH
export CVS_TARBALL_STASH="http://oesources.org/sources/current/"
if [ "$PS1" ]; then
    if [ "$BASH" ]; then
        export PS1="\[\033[01;32m\]OE:$MY_OE_CONF\[\033[00m\] ${PS1}"
    fi
    fi home/oe/beagleboard/beagleboard/conf/local.conf: DISTRO = "angstrom-2008.1"
BBFILES = "/home/saul/oe/openembedded.org/openembedded/recipes/*/*.bb"
TMPDIR = "/home/saul/oe/tmp"
MACHINE = "beagleboard" ENABLE_BINARY_LOCALE_GENERATION = "0"
```

Obtendo uma imagem linux com o OE:

```
$ source profile.sh \\ ativa as configurações relativas à beagleboard
$ bitbake -b console-image
```

Anexo E

Configuração da porta serial (Serial Port Setup):

```
$ minicom -s
```

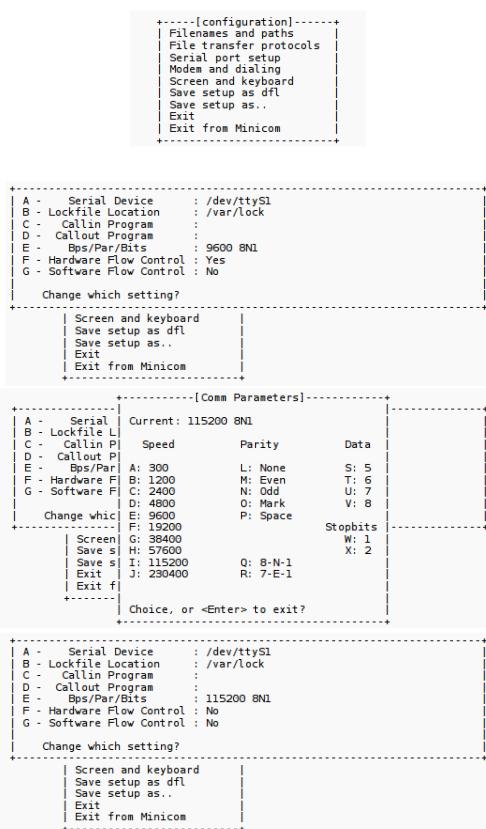
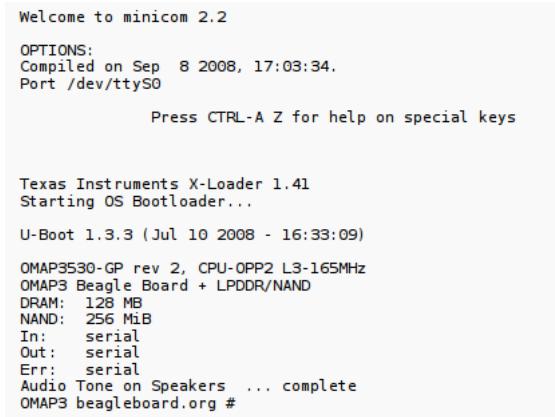


Figura 4.2: Configurando o *minicom* para acesso à BB

ttySX diz respeito a que porta serial está conectada a BB (X=0,1...) no caso de adaptadores USB-SERIAL costuma-se utilizar ttyUSB0.

Ao ligar a BB, a seguinte tela deve ser exibida:



```
Welcome to minicom 2.2

OPTIONS:
Compiled on Sep  8 2008, 17:03:34.
Port /dev/ttyS0

Press CTRL-A Z for help on special keys

Texas Instruments X-Loader 1.41
Starting OS Bootloader...

U-Boot 1.3.3 (Jul 10 2008 - 16:33:09)

OMAP3530-GP rev 2, CPU-OPP2 L3-165MHz
OMAP3 Beagle Board + LPDDR/NAND
DRAM: 128 MB
NAND: 256 MiB
In:   serial
Out:  serial
Err:  serial
Audio Tone on Speakers ... complete
OMAP3 beagleboard.org #
```

Figura 4.3: Tela da BB já no segundo estágio de inicialização (u-boot)

Os comando abaixo, são referentes às configurações necessárias ao boot ser dado pelo cartão SD e configurações de vídeo:

```
# setenv bootargs 'console=ttyS2,115200n8 console=tty0 root=/dev/mmcblk0p2 rw rootfstype=ext3 root-
wait omapfb.video_mode=1024x768MR-16@60'
# setenv bootcmd 'mmcinit; fatload mmc 0 0x80300000 uImage; bootm 0x80300000'
# saveenv
```

- “# mmcinit” - viabiliza o acesso ao cartão SD (dependendo da versão do u-boot pode ser encontrado o comando “mmc init”).
- “# fatload mmc 0 0x80300000 uImage” - traz para a posição de memória livre 0x80300000 da beagle board o arquivo “uImage” localizado no cartão de memória.
- “# bootm 0x80300000” - inicializa o sistema operacional que utiliza o cartão SD através do micro Kernel copiado para a memória da beagle board.

O primeiro boot demorou mais de 15 minutos, provavelmente por algumas alterações efetuadas no micro kernel, além de, claro, ocorrer a descompactação do próprio micro kernel.

Anexo F

Formatando o Cartão SD através da ferramenta *fdisk*

Primeiro foi necessário descobrir qual dispositivo estava relacionado com o cartão SD, tarefa simples com o uso do comando: \$ dmesg | tail

saída:

...

```
[ 6854.215650] sd 7:0:0:0: [sdc] Mode Sense: 0b 00 00 08
[ 6854.215653] sd 7:0:0:0: [sdc] Assuming drive cache: write
through
[ 6854.215659] sdc: sdc1
[ 6854.218079] sd 7:0:0:0: [sdc] Attached SCSI removable disk
[ 6854.218135] sd 7:0:0:0: Attached scsi generic sg2 type 0
...
$ sudo fdisk /dev/sdc
```

Deleta-se a partição atual

Command (m for help): [d]

Entra-se no modo *expert*

Command (m for help): [x]

Altera-se o número de cabeças

Expert Command (m for help): [h]

Number of heads (1-256, default xxx): [255]

Altera-se o número de setores

Expert Command (m for help): [s]

Number of sectors (1-63, default xxx): [63]

o número de cilindros foi obtido da seguinte maneira: “capacidade do SD (em bytes)/255/63/512”
(truncado)

Expert Command (m for help): [c]

Number of cylinders (1-256, default xxx): [245]

Deixa-se o modo *expert*

Expert Command (m for help): [r]

Cria-se uma nova partição

Command (m for help): [n]

Command action e extended p primary partition (1-4)

[p]

Partition number (1-4): [1]

First cylinder (1-245, default 1): [<pressione enter>]

Last cylinder or +size or +sizeM or +sizeK (1-245, default 245): [+50]

Marca-se a partição com compatibilidade para DOS

Command (m for help): [t]

Hex code (type L to list codes): [c]

Marca-se a partição como *bootável*

Command (m for help): [a]

Partition number (1-4): [1]

Cria-se a partição do sistema de arquivos

Command (m for help): [n]

Command action e extended p primary partition (1-4) [p]

Partition number (1-4): [2] First cylinder (52-245, default 52): [<Pressione Enter>]

Last cylinder or +size or +sizeM or +sizeK (52-245, default 245): [<Pressione Enter>]

Command (m for help): [w]

já fora do *fdisk*, formata-se cada uma das partições:

```
$ sudo mkfs.vfat -n LABEL1 /dev/sdc1
```

```
$ sudo mkfs.ext3 -L LABEL2 /dev/sdc2
```

Anexo G

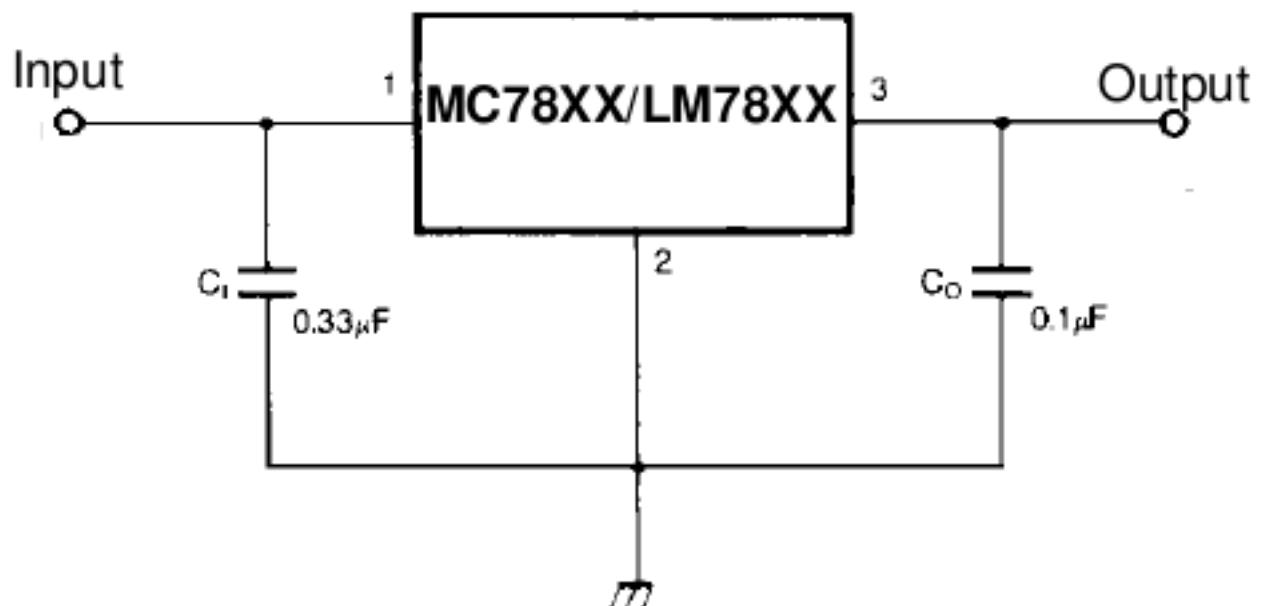


Figura 4.4: Aplicação do CI LM7805 como regulador de tensão.

Referências Bibliográficas

- [1] Angstrom-Linux (2010). <<http://www.angstrom-linux.org>> acesso em: 15/05/2010.
- [2] Axelson, J. (2003). *Embedded ETHERNET and INTERNET Complete Designing and Programming Small Devices for Networking*. Lakeview Research LLC.
- [3] Baron, R. J. and Higbie, L. (1992). *Computer Architecture*. Addison-Wesley Publishing Company.
- [4] CPQD (2010). <<http://www.cpqd.com.br/joomladev/pad-e-inovacao/4530-tecnologias-para-rede-optica-convergente.html>> acesso em: 15/05/2010.
- [5] Hennessy, J. L. and Patterson, D. A. (2003). *Computer Architecture, 3ed.* Morgan Kaufmann Publishers.
- [6] IEEE (2005). *INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERING - Standard 802.3*.
- [7] LM7805 (2010). Datasheet <<http://www.fairchildsemi.com/ds/lm>> acesso em: 12/05/2010.
- [8] Minoli, D. (2005). *A Networking Approach to GRID Computing*. John Wiley and Sons.
- [9] OpenEmbedded (2010). <<http://www.openembedded.org>> acesso em: 15/05/2010.
- [10] Tanenbaum, A. S. (1996). *Computer Networks, 3ed.* Prentice Hall.
- [11] Tanenbaum, A. S. (2008). *Modern Operating Systems*. Pearson Prentice Hall.
- [12] TexasInstruments (2010). Omap3 <<http://focus.ti.com/docs/prod/folders/print/omap3530.html>> acesso em: 15/05/2010.
- [13] Wikipedia (2010a). Arm <http://en.wikipedia.org/wiki/arm_architecture> acesso em: 15/05/2010.
- [14] Wikipedia (2010b). Mpi <http://en.wikipedia.org/wiki/message_passing_interface> acesso em: 10/05/2010.
- [15] Wikipedia (2010c). Usb <http://en.wikipedia.org/wiki/universal_serial_bus> acesso em: 15/05/2010.