A programação paralela: novos desafios para novas arquiteturas

Nicolas Maillard
Instituto de Informática – UFRGS
18 de Outubro de 2007

GPPD

Roteiro da apresentação

- O contexto: do Processamento de Alto Desempenho à programação paralela...
  - Definição,
  - Plataformas computacionais,
  - Tendências arquiteturais.
- O problema: a programação tradicional continua sendo seqüencial!
  - Quais alternativas?
  - Quais limitações nas alternativas?
- A pesquisa na área
  - Quais rumos possíveis nos x próximos anos?

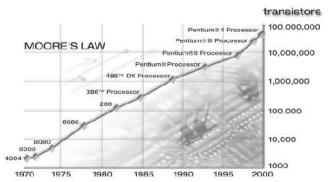
Processamento de Alto Desempenho = Fórmula 1

- Quer-se executar um programa o mais eficientemente possível.
- Para ganhar a corrida, vale tudo:
  - Hardware (CPU & Rede),
  - Sistema operacional,
  - Compilador,
  - Escrita do código,
  - Algoritmos,
  - Ver - http://www.sbc.org.br/sbac/2007/marathon.php
- Interessa todas as áreas !
- Qual "carro" é melhor ? O caro, super-sofisticado ou o barato muito disponível?
- Nem todo o mundo precisa de um carro de fórmula 1.

Por que o alto desempenho?

Lei de Moore…
Basta esperar!

Mas nem sempre se pode esperar!



- Simulação nuclear
- Procura de poços de petróleo
- Realidade virtual

Projetos de Alto Desempenho

- Projetos pesados,
- Projetos estratégicos,
- Projetos internacionais (Globus, Earth Simulator, Datagrid),
- Projetos caros,
- Parcerias com industriais importantes,
- … mas isso não significa que o Brasil esteja por fora!
  - Petrobrás, CPTEC, Banco do Brasil, Correios, Infraero, ... são casos de sucesso no uso de PAD.

Máquinas de Alto desempenho

- Nos anos 70s: processadores vetoriais, pipe-line
- Problemas de tempo de acesso:
  - Velocidade do Processador > Tempo de acesso à memória
  - Tempo de acesso à memória > tempo acesso rede
  - tempo acesso rede > Tempo de acesso disco
- Soluções:
  - CACHE
  - Uso do paralelismo: mais de um processador trabalha sobre um problema

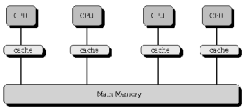
Máquinas Dedicadas (MPP)

- Processadores alto-desempenho (alpha, RS6000...)
- Rede dedicada
- SW proprietário

- Centenas de nós
- IBM, Cray (SGI), HP, DELL, ITAUTEC, Sun, BULL...
- Manutenção e instalação complexas
- Usadas em centros nacionais

Máquinas com Memória Compartilhada

- Todos os processadores acessam uma memória única
- Caro, mas bom custo/facilidade de uso!
- Algumas dezenas de procs.
- Gargalo = barramento!
- Facilita a programação e a distribuição dos cálculos



Agregados (Clusters)

CPU baratas (?) – Pentium IV, Athlon, Opteron – Itanium – Alpha – RS6000, …

+ Rede barata (?) Ethernet 100, Gbit Ethernet, SCI, Myrinet, infiniband…

+ Sistema Operacional "adequado" (sistema de arquivos compartilhado, clonagem de SO em cada nó…)

= Máquina Agregada

- Milhares de nós
- "Barato"
- "Caseiro" (Linux, Ethernet, procs. "de prateleira")

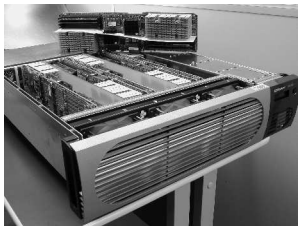
Alguns Cluster no RS

- UFRGS
  - Cluster Dell (Labtec) 20 nós dual PIII, fast Ethernet
  - Cluster Itautec

- Santa Maria:
  - 12 nós Pentium III a 1GHz bi-processados
  - memória principal de 1 GBytes
  - rede Gigabit Ethernet.

Máquinas híbridas

- Misturam memória compartilhada e distribuída.
- Exemplo: o Cray XD1 (UFRGS)



Grids

Novo paradigma

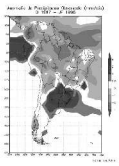
"A Computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities." [Foster:99]

- Analogia poder computacional/poder elétrico
- Serviços pela Web
- "Internet Computing"

Projetos de Grids no Brasil

- LNCC + CENAPADS
- Walfredo Cirne: MyGrid / OurGrid
  http://www.ourgrid.org/mygrid

- LNCC, Bruno Schulze
  http://www.lncc.br/~schulze/



- Projeto FINEP GBRAMS (UFRGS/INPE/USP)
- Grid nacional?

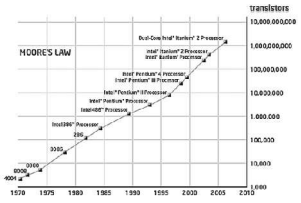
… e todo o mais

- Uso de ciclos ociosos (Condor)
- Problem solving environments (Netsolve)
- Peer-to-Peer computing (Seti@home)
- Ubiquous Computing
- Global computing
- Pervasive Computing
- Etc…

A tendência na arquitetura

- A lei de Moore continua...
- Mas o consumo de energia se torna (muito) preocupante!
- Devido:
  - à densidade
  - À freqüência de relógio
- Solução:

MULTICORE



Vários níveis de paralelismo

Começando no nível continental:
a grade computacional...

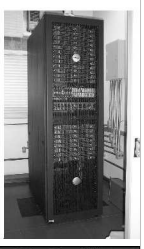
Vários níveis de paralelismo

... Passando pelo nível nacional...

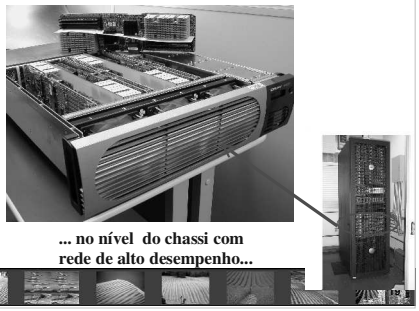
... A constelação...

Vários níveis de paralelismo

... Em nível local:
Máquina dedicada/
Cluster...

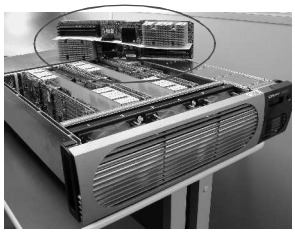


Vários níveis de paralelismo



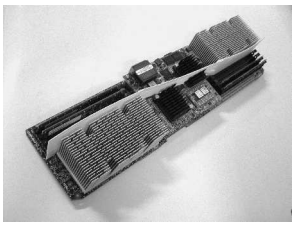
... no nível do chassi com rede de alto desempenho...

Vários níveis de paralelismo



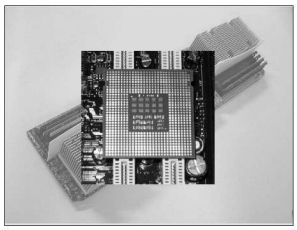
No nível do nó multi-processado

Vários níveis de paralelismo



No nível do nó multi-processado...

Vários níveis de paralelismo



... Até o processador multi-core!

Qual é o problema?

- O gerenciamento de tais plataformas é complicado.
  - Bootar leva tempo!
  - Acesso, segurança, monitoração, instalação de SW...
  - Depuração de programas!
  - Portabilidade dos programas!
  - Ciclo de vida dos programas!
- A **programação** de tais plataformas é complicada!

O desafio da programação paralela

- Em todos os níveis, as arquiteturas provêm paralelismo importante.
- Linguagens de programação / algoritmos não são projetados para aproveitar este paralelismo
  - Algoritmos: o modelo de referência é a máquina de Turing...
  - Modelos paralelos não capturam todos os parâmetros ou são impraticáveis
    - PRAM (memória compartilhada infinita)
    - BSP / CSM (fortemente síncrono)
    - LogP (poucos resultados)

Linguagens para programação paralela

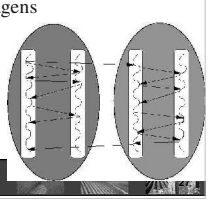
- Abordagens para sua programação:
- Fornecer uma abstração de alto-nível, tal como o objeto (Java/Corba)
  - Facilita a migração e o desenvolvimento;
  - Quid da performance?
- Deixar o programador definir vários fluxos de execução:
  - Com comunicação por memória (virtualmente) compartilhada – Threads
  - Com comunicação por rede (Message Passing)
- Fazer com que o compilador resolva...
  - Paralelização automática

Abordagem 1 - Objetos Distribuídos

- Virtualização do HW/O.S. através de uma Máquina Virtual;
- Encapsulação de dados/métodos no Objeto;
- Publicação / descoberta dos métodos;
- Invocação remota
  - Dados de E/S a serem trocados devem ser serializados!
- Facilita a portabilidade/ubiquidade... Mas com sobrecusto grande!
  - Adaptado para Ubiquous Computing, Pervasive Computing, ...
  - Exemplos: projeto Ibis (H. Bal/Amsterdã); ProActive (INRIA)

Abordagem 2 - Processos comunicantes

- Extensões de linguagens seqüenciais clássicas:
  - Bibliotecas de threads
    - Definição, criação e sincronização de fluxos de execução.
  - Bibliotecas de troca de mensagens
    - PVM, MPI.



Exemplo de código MPI

Tipo definido pelo MPI

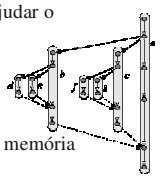
Emissão/recepção de mensagem

```
void  main() {
  int p, r, tag = 103;
  MPI_Status stat;
  double valor ;
  MPI_Init(&argc, &argv) ;
  MPI_Comm_rank(MPI_COMM_WORLD, &r);
  if (r==0) {
     printf("Processor 0 sends a message to 1\n");
     valor = 3.14 ;
     MPI_Send(&valor, 1, MPI_DOUBLE, 1, tag,
    MPI_COMM_WORLD);   }
 else {
     printf("Processor 1 receives a message from 0\n");
     MPI_Recv(&valor, 1, MPI_DOUBLE, 0, tag, MPI_COMM_WORLD, &stat);

     printf("O valor recebido vale %.2lf\n", valor);   }
}
```


Abordagem 3 - Paralelização automática

- Paralelização direta e automática: muito **difícil**.
- **Solução**: anotar o código fonte para ajudar o compilador a extrair o paralelismo.
  - OpenMP (estático)
  - Cilk (dinâmico)
- Problema: hoje, só funciona bem com memória compartilhada...



Exemplo de código OpenMP

Sequential code	Parallel Open-MP code
double res[10000];	double res[10000];
	#pragma omp parallel for
for (i=0 ; i<10000; i++)	for (i=0 ; i<10000; i++)
compute(res[i]);	compute(res[i]);

Conclusão do estado da Arte

- Não existe uma interface única de programação paralela.
- Existem soluções para memória compartilhada e distribuída:
  - MPI / Open-MP
  - Fabricantes incentivam o uso das mesmas
- O que existe não dá conta de:
  - Dinamicidade
  - Heterogeneidade
- A pesquisa está focando esses dois tópicos
  - MPI-2, Cilk, Charm++, Satin/Ibis, Nesl, UPC, etc...
  - A pesquisa (muito) "aplicada" limitou o tratamento do problema ao paralelismo trivial.

Abordagens na pesquisa para enfrentar o problema...

- Algoritmos: ir além do "Task Farm"
  - **Divisão & Conquista**, para criação "sob-demanda" do paralelismo.
  - Necessita controle da recursividade/granularidade!
- Linguagem: dar suporte para ambientes distribuídos
  - Parecido ao Open-MP/Cilk, mas com **troca de mensagens**.
- **Compilação on-line** para gerar código eficiente adaptado à máquina alvo
  - Threads vs. Troca de mensagens

Linguagens: suporte ao D&C com mensagens

- Linguagens com novas instruções:
  - Criação de fluxos de execuções (tarefas)
    - Spawn, Fork (como no Cilk)
  - Sincronização / troca de dados
    - Novidade para levar em consideração comunicações em rede, além de memória compartilhada.
      - Semelhante ao "pipe" Unix.
    - Cilk/Open-MP apenas provê acessos concorrentes a dados em memória compartilhada!
    - Levar em consideração a rede possibilita melhor controle de escalonamento.

Conclusões

- A programação paralela está presente hoje em qualquer ambiente computacional
  - Até o processador é paralelo.
- Tem gente dizendo que uma empresa que não se conscientizar disso hoje terá o mesmo destino que as firmas que negligenciaram o computador nos anos 60.
- Profissionais devem ser preparados por isso.
  - Threads, OpenMP, MPI são ferramentas básicas.
- Como adaptar o ensino?

Conclusões (2)

- Plataformas de Grids já existem, ou estão sendo construídas.
  - Apenas para aplicações trivialmente paralelas.
- As linguagens de programação estão evoluindo para acompanhar Grids e Arquiteturas Multicores
  - MPI-2, UPC (Berkeley), ...

- Deve-se prover soluções em novas linguagens para programação eficiente das plataformas da década 2010.

GPPD

Boa sorte!

Nicolas Maillard

Informática UFRGS

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL