

UFRGS
GPPD

Clusters de Alto Desempenho e programação distribuída

Nicolas Maillard
Instituto de Informática – UFRGS
2008

nicolas@inf.ufrgs.br

Informática UFRGS

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Roteiro da apresentação

- O contexto: do Processamento de Alto Desempenho à programação paralela...
 - Definição,
 - Plataformas computacionais,
- Clusters:
 - Caraterísticas
 - Gerenciamento
- Programação paralela em Clusters: MPI!
 - Quais alternativas?
 - Message Passing Interface

nicolas@inf.ufrgs.br

GPPD

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,
- 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.

nicolas@inf.ufrgs.br

GPPD

Computação de Alto Desempenho

- Quando usar?** Problemas “difíceis” necessitam de alto desempenho (ciência, engenharia e negócios)
- Por que usar?** Pressões do mercado pedem ciclo de inovação acelerado e redução de custo
 - Design de aeronaves utilizando materiais compostos
 - Melhoria na eficiência e segurança dos combustíveis de veículos
 - Simulação da catálise de enzimas e proteínas
 - Design de medicamentos
 - Simulação de dispositivos eletrônicos em escala nano
 - Modelagem de risco em portfólios financeiros
 - Criação e aprimoramento de conteúdo digital
 - Modelagem e otimização da cadeia de suprimentos
 - Previsão do clima

nicolas@inf.ufrgs.br

GPPD

Por que o alto desempenho?

Lei de Moore...
Basta esperar!

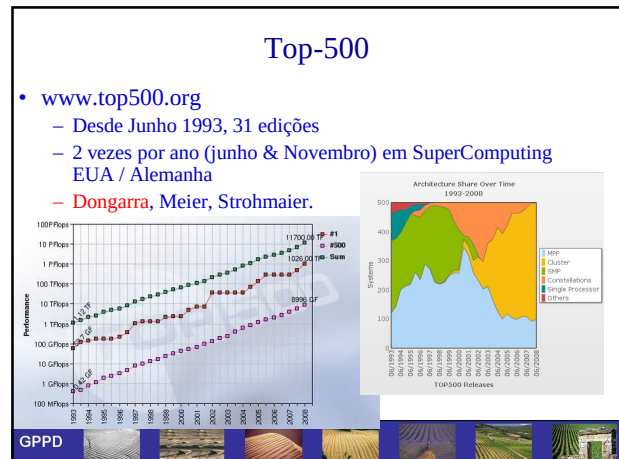
nicolas@inf.ufrgs.br

GPPD

Supercomputadores Pessoais?

	1991	1998	2005
Sistema	Cray Y-MP C916	Sun HPC10000	Shuttle @ NewEgg.com
Arquitetura	16 x Vector 4GB, Bus	24 x 333MHz Ultra-SPARCII, 24GB, SBus	4 x 2.2GHz x64 4GB, GigE
SO	UNICOS	Solaris 2.5.1	Windows Server 2003 SP1
Performance	~10 GFlops	~10 GFlops	~10 GFlops
Top500 #	1	500	N/A
Preço	\$40,000,000	\$1,000,000 (40x drop)	< \$4,000 (250x drop)
Clientes	Lab. Governo	Grandes Empresas	Engenheiros e Cientistas
Aplicações	Secretas, Clima, Pesquisa em Física	Manufatura, Energia, Finanças, Telecom	Bioinformática, Ciências, Mídia Digital

GPPD



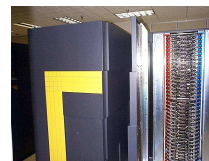
Tecnologia de rede no top5000 2008/1

Interconnect Family	Count	Share %	Rmax Sum (GF)	Rpeak Sum (GF)	Processor Sum
Myrinet	12	2.40 %	341054	493773	56740
Quadrics	5	1.00 %	112726	138342	21232
Gigabit Ethernet	284	56.80 %	3492978	6654720	680012
Infiniband	121	24.20 %	4349473	5956890	592978
Crossbar	3	0.60 %	61165	73629	8694
Mixed	2	0.40 %	76667	100866	15504
NUMalink	8	1.60 %	180613	201103	31360
SP Switch	17	3.40 %	292602	369081	45136
Proprietary	40	8.00 %	2417348	3004604	874396
Cray Interconnect	8	1.60 %	375391	452371	88167
Totals	500	100 %	11700016.97	17445379.70	2414219

nicolas@inf.ufrgs.br



Máquinas Dedicadas (MPP) – década 80-90



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

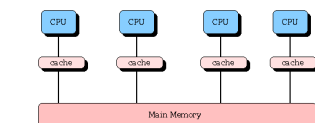
- Centenas de nós
- IBM, Cray (SGI), HP, DELL, ITAUTEC, Sun, BULL...
- Manutenção e instalação complexas
- Usadas em centros nacionais

nicolas@inf.ufrgs.br



Máquinas com Memória Compartilhada – 90-95

- 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



nicolas@inf.ufrgs.br



Agregados (Clusters) – 95 - hoje

- 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”)

nicolas@inf.ufrgs.br



Alguns Cluster no RS



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

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

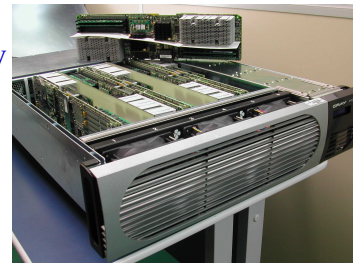


nicolas@inf.ufrgs.br

GPPD

Máquinas híbridas

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



nicolas@inf.ufrgs.br

GPPD

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”

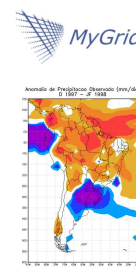


nicolas@inf.ufrgs.br

GPPD

Projetos de Grids no Brasil

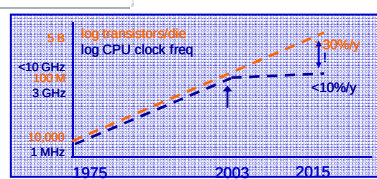
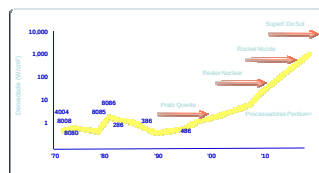
- LNCC + CENAPADS
- Walfredo Cime: MyGrid / OurGrid
<http://www.ourgrid.org/mygrid>
- LNCC, Bruno Schulze
<http://www.lncc.br/~schulze/>
- Projeto FINEP GBRAMS (UFRGS/INPE/USP)
- Grid nacional?



nicolas@inf.ufrgs.br

GPPD

Bem Vindo à era Multi-Core



nicolas@inf.ufrgs.br

GPPD

... e todo o mais

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

nicolas@inf.ufrgs.br

GPPD

Vários níveis de paralelismo



Começando no nível continental:
a grade computacional...

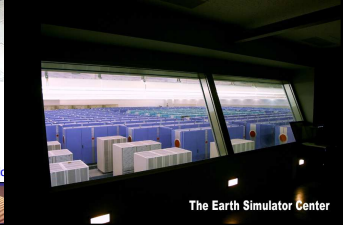
nicolas@inf.ufrgs.br

GPPD

Vários níveis de paralelismo



... Passando pelo nível nacional...



... A constelação...

nic

GPPD

Vários níveis de paralelismo



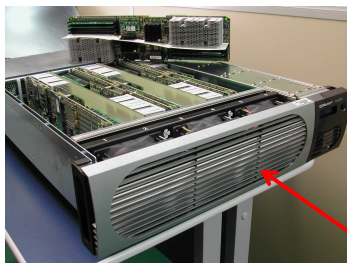
... Em nível local:
Máquina dedicada/
Cluster...



nicolas@inf.ufrgs.br

GPPD

Vários níveis de paralelismo

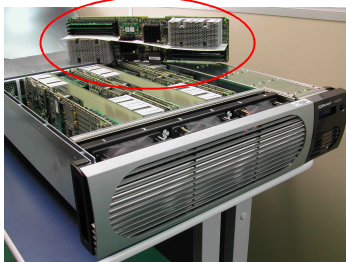


... no nível do chassi com
rede de alto desempenho...



GPPD

Vários níveis de paralelismo

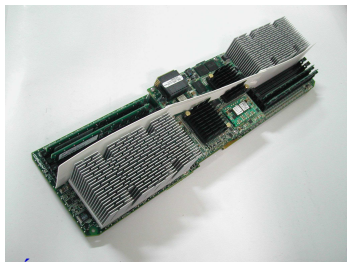


No nível do nó
multi-processado...

nicolas@inf.ufrgs.br

GPPD

Vários níveis de paralelismo



No nível do nó
multi-processado...

nicolas@inf.ufrgs.br

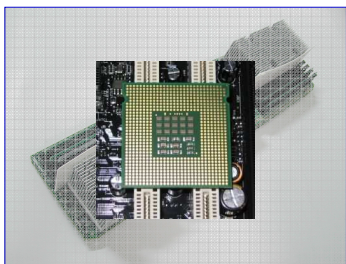
GPPD

Slide 19

- NM1** A grade já é realidade:
- grid 5000
 - datagrid / CERN
 - grade japonesa
 - PAUÁ
 - LNCC
 - GBRAMS

Nicolas Maillard; 17/08/2006

Vários níveis de paralelismo



... Até o processador multi-core!

nicolas@inf.ufrgs.br

GPPD

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!

nicolas@inf.ufrgs.br

GPPD

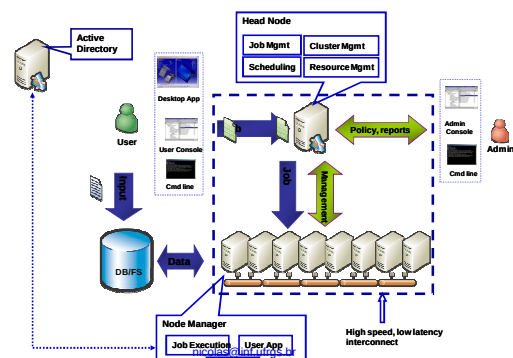
Cluster - arquitetura

- Um cluster = um conjunto de nós
 - PC, processadores dedicados...
- Tipicamente, um nó especial serve de **front-end**
 - Interface com os usuários,
 - Servidora de disco,
 - Controle do acesso aos outros nós,
 - Servidora HTTP,
 - Compilação dos programas...
- Os outros (muitos) nós servem para computação
 - Configuração enxuta do núcleo do Sis. Op.,
 - Não acessíveis diretamente pelos usuários,
 - Possivelmente usam uma interconexão específica (Myrinet, GB Ethernet, SCI, infiniband...)
 - Uso em exclusão mútua.

nicolas@inf.ufrgs.br

GPPD

Exemplo: Microsoft Compute Cluster Solution



GPPD

28

Instalação de um cluster

1. Instala (linux, por exemplo) no front-end.
 1. Criar um /home e os logins dos usuários,
 2. Instalar os compiladores, sshd, nfsd, e um software de controle de acesso.
2. Instala um nó.
3. Replica a imagem do nó.
4. Atribui IPs aos nós.
 1. Conecta ao front-end.
5. Exporta /home nos nós.



nicolas@inf.ufrgs.br

GPPD

Serviços especiais

- Compiladores / programação:
 - Bibliotecas para usuários de HPC
 - BLAS, Lapack, MPI, OpenMP, boost, MKL...
- Sistema de **controle de acesso/batches**
 - OpenPBS, Sun SGE, OAR, ...
 - Provê uma interface para:
 - Reservar n nós de cálculo para uso em exclusão mútua.
 - Quantos, quais, quando, durante quanto tempo?
 - Trata as filas de requisições de nós.
 - Garante o acesso aos nós e a execução (em modo batch).
 - Provê mecanismos de acompanhamento e de logs.

nicolas@inf.ufrgs.br

GPPD

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)

nicolas@inf.ufrgs.br

GPPD



Linguagens para programação paralela

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

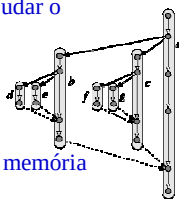
nicolas@inf.ufrgs.br

GPPD



Abordagem 1 - 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...



nicolas@inf.ufrgs.br

GPPD



Exemplo de código OpenMP

Sequential code

```
double res[10000];

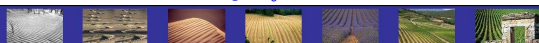
for (i=0 ; i<10000; i++)
  compute(res[i]);
```

Parallel Open-MP code

```
double res[10000];
#pragma omp parallel for
for (i=0 ; i<10000; i++)
  compute(res[i]);
```

nicolas@inf.ufrgs.br

GPPD



Abordagem 2 - 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 Ubiquitous Computing, Pervasive Computing, ...
 - Exemplos: projeto Ibis (H. Bal/Amsterdã); ProActive (INRIA)



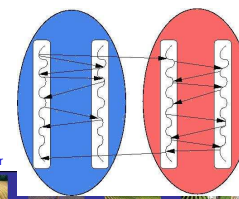
nicolas@inf.ufrgs.br

GPPD



Abordagem 3 - Processos comunicantes

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



nicolas@inf.ufrgs.br

GPPD



Message Passing Interface

nicolas@inf.ufrgs.br

GPPD

Plano

- Introdução a Message Passing Interface
 - O modelo de programação paralela suportado pelo MPI.
 - “MPI for Dummies”: as 6 instruções básicas.
 - MPI avançado:
 - Comunicações não-bloqueantes
 - Comunicações coletivas
 - Tipos de dados definidos pelo usuário
 - MPI-2
 - MPI-IO
 - Criação dinâmica de processos.
 - Que mais?

nicolas@inf.ufrgs.br

GPPD

Algumas referências sobre MPI

- <http://www.mpi-forum.org>, para a norma MPI.
- Conferência internacional: EuroPVM-MPI (LNCS, Springer)
- Gropp, William *et al.*, **Using MPI**, MIT Press.
- Gropp, William *et al.*, **Using MPI-2**, MIT Press.
- Snir, M. *et al.*, Dongarra, J., **MPI: The Complete Reference**.

nicolas@inf.ufrgs.br

GPPD

O que é um programa MPI?

- Um programa MPI = um conjunto de **p** processos “pesados” que se comunicam através de **mensagens**.
 - Processo pesado = **memória local** e privada,
 - Mensagens = rede (Bom... depende...),
 - Tipicamente, os p processos são disparados no início.
- MPI é uma **biblioteca**, não uma L.P.
- SPMD - Single Program, Multiple Data approach
 - Programa uma vez, mas vários processos executam!
 - Cada processo é identificado por um **rank**.
- Em tese: escreve-se um programa C, fork-a-se p processos e os clones usam sockets.
 - Quem quer programar assim mesmo?

nicolas@inf.ufrgs.br

GPPD

O caso de sucesso do MPI

- Antes do MPI, tinha o PVM (Parallel Virtual Machine)
 - Era na época das primeiras NoWs.
 - Útil, mas faltava padronização.
- 1995 — MPI 1.1
 - 1997 — MPI 1.2
 - 1998 — MPI-2
 - 2000+ — MPI-2 fornecido nas principais distribuições.
- 2 principais distribuições Open Source:
 - OpenMPI (www.open-mpi.org),
 - MPI-CH (www-unix.mcs.anl.gov/mpi/mpich/).
- Muitas distribuições comerciais.
- É fato que a ampla maioria das aplicações executando em Clusters são baseadas no MPI.
- “MPI is as simple as using 6 functions and as complicated as a user wishes to make it.” [B. Gropp]

nicolas@inf.ufrgs.br

GPPD

MPI em 6 palavras

- 1) **MPI_Init**(&argc, &argv) // No comment.
- 2) **MPI_Comm_rank**(&r, communicator)
 - // retorna o rank in na variável int ‘r’
- 3) **MPI_Comm_size**(&p, communicator)
 - // retorna o número de processos na variável int ‘p’
- 4) **MPI_Send**(/* muitos args */)
- 5) **MPI_Recv**(/* quase tantos args */)
- 6) **MPI_Finalize**() // No comment

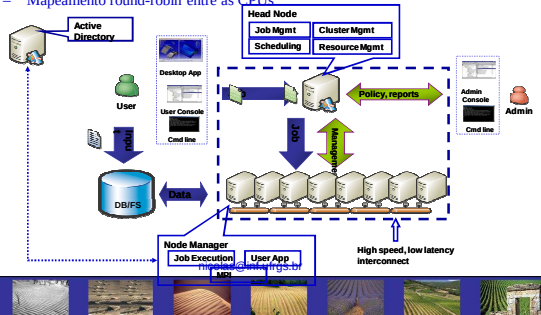
O communicator básico é a constante
MPI_COMM_WORLD.

nicolas@inf.ufrgs.br

GPPD

Executando um programa MPI

- `mpirun -np 10 /path/to/mpi-program`
- Um arquivo textual descreve os endereços IP das CPUs que vão executar os processos.
 - Mapeamento round-robin entre as CPUs



GPPD

Como é uma mensagem MPI?

- Perfil:
`int MPI_Send(void*, int, MPI_Datatype, int, int, MPI_Comm).`
- Chamada típica:
`MPI_Send(&work, 1, MPI_INT, dest, WORKTAG, MPI_COMM_WORLD);`
- Manda o conteúdo de um buffer do processo atual para um destinatário.
- O **buffer** é definido pelos 3 primeiros argumentos:
 - `Work (void*)`: ponteiro para a área onde se encontram os dados.
 - `1, MPI_INT`: número e tipo básicos dos elementos no buffer (quase `sizeof()`!)
- A mensagem é identificada por um tag (ver `WORKTAG`).
 - Deve fechar o tag entre o `Send` e o `Recv`.
- O processo destinatário é identificado por seu rank, dentro de um comunicador.

nicolas@inf.ufrgs.br

GPPD

Do lado do destinatário....

- Perfil:
`int MPI_Recv(void*, int, MPI_Datatype, int, int, MPI_Comm, MPI_Status*).`
- Chamada típica:
`MPI_Recv(&result, 1, MPI_DOUBLE, 1, tag, MPI_COMM_WORLD, &status);`
- Recebe o conteúdo de uma mensagem e o armazena no buffer 'result'.
- O buffer de recepção é especificado assim como no `Send` (3 primeiros args).
- Tem um rank da fonte, e um tag.
 - Pode-se usar **corrinhas**: `MPI_ANY_SOURCE`, `MPI_ANY_TAG`.
- O argumento 'status' (o último) provê informação a respeito da mensagem que foi recebida.

nicolas@inf.ufrgs.br

GPPD

Exemplo básico e completo.

```
void main() {
    int p, r, tag = 103;
    MPI_Status stat;
    double val;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &r);
    if (r==0) {
        printf("Processor 0 sends a message to 1\n");
        val = 3.14;
        MPI_Send(&val, 1, MPI_DOUBLE, 1, tag, MPI_COMM_WORLD);
    }
    else {
        printf("Processor 1 receives a message from 0\n");
        MPI_Recv(&val, 1, MPI_DOUBLE, 0, tag, MPI_COMM_WORLD,
            &stat);
        printf("I received the value: %.2lf\n", valor);
    }
}
```

Tipos abstratos do MPI

nicolas@inf.ufrgs.br

GPPD

Algumas observações

- **MPI não se limita a programas Mestre/Escravo**
 - Padrão frequente: processos de rank par fazem alguma coisa, os ímpares fazem outra coisa.
- Os tags devem ser predefinidos pelo programador.
 - O conjunto (source, tag, dest) identifica a mensagem, então cuidado com possíveis confusões (casts).
- O buffer é uma área de memória **contígua**
 - Deve-se serializar dados não-contíguos para comunicá-los.
- A mensagem tem tamanho fixo, e o recipiente deve conhecê-lo para efetuar o `Recv`.
 - Possivelmente, se deve mandar primeiro o tamanho, e depois os dados.

nicolas@inf.ufrgs.br

GPPD

Comm. Bloqueantes vs. Não-bloqueantes

- `MPI_Recv(&x,...)` é bloqueante.
 - Quando se executa a próxima instrução, `x` já tem o valor correto.
- `MPI_Send(&x,...)` é não-bloqueante
 - Porém, `x` é bufferizado, e por isso a próxima instrução pode alterar `x` sem problema.
 - `Send` é não-bloqueante... Até estourar o buffer interno!
- Variações não-bloqueantes: `MPI_Irecv()` e `MPI_Isend()`
 - Mesmos argumentos como `Recv/Send`, com um extra de tipo `MPI_Request`.
 - O `MPI_Request` possibilita testar a compleção de comunicações não-bloqueantes.
- Existe também versões explicitamente bufferizadas do `Send/Recv`: `MPI_Bsend()`.

nicolas@inf.ufrgs.br

GPPD

Testar e esperar comm. não bloqueantes

- **MPI_Test**(MPI_Request* req, int* flag, MPI_Status* stat)
 - Seta 'flag' a 0 ou 1, em função do termino da com não-bloqueante descrita por 'req'.
 - Deve-se testar 'flag' depois...
- **MPI_Wait**(MPI_Request* req, MPI_Status* stat)
 - Bloqueia até a comunicação não-bloqueante tenha terminado.
- Prática (muito) comum (sobreposição computação/comunicação) :
 - Dispara uma comm não-bloqueante (e.g recv),
 - (em um laço) executa todo o que se pode fazer sem ter recebido os dados, testando a recepção.
 - Se o laço acabar, bloqueia com **MPI_Wait**

GPPD



Comunicações coletivas

- Um dos pontos fortes do MPI: todas as comunicações coletivas são **padronizadas**.
 - Cabe à implementação da biblioteca otimizá-las com árvores de broadcast, roteamento em hyper-cube, etc...
 - Tão logo se usa mais de 2 processos numa comunicação, nunca use Send/recv!
 - Não reinvente a roda...
- Por definição, todos os processos se sincronizam quando efetuam uma com. coletiva.
 - O MPI não provê versões não-bloqueantes.
- Pode-se particionar MPI_COMM_WORLD em sub-communicators, para possibilitar comunicações dentro de um sub-grupo de processos.

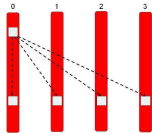
nicolas@inf.ufrgs.br

GPPD



Algumas comm. coletivas: Bcast

- `int root = 0;`
`double dados[ n ]`
MPI_Bcast(dados, n, MPI_DOUBLE, root, MPI_COMM_WORLD);
- O processo de rank 'root' manda n doubles (em dados) para todos os demais.
- TODOS os processos chamam MPI_Bcast
 - Há sincronização global,
 - Internamente, 'root' vai mandar mensagens, os p-1 outros processos vão receber.
 - O buffer ('dados') tem uma **semântica diferenciada** em função do rank.



GPPD



MPI_Reduce

- Perfil:


```
MPI_Reduce(void*, void*, int, MPI_Datatype, MPI_Op,
            int, MPI_Comm)
```
- Chamada típica:


```
double* sum_buf, res;
sum_buf = malloc(10*sizeof(double));
MPI_Reduce(sum_buf, res, 10, MPI_DOUBLE, MPI_SUM,
            root, MPI_COMM_WORLD);
```
- Aplica um operador de redução no buffer, manda os resultados parciais para 'root', que recupera o resultado global em 'res'.
- O operador pode ser MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, MPI_LAND, MPI_LOR, etc. . .

nicolas@inf.ufrgs.br

GPPD



Calculando Pi (1)

```
#include "mpi.h"
#include <math.h>
int main(int argc, char *argv[])
{
    int done = 0, n, myid, numprocs, i, rc;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x, a;

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

    while (!done) {
        if (myid == 0) {
            printf("Enter the number of intervals:
                  (0 quits) ");
            scanf("%d", &n);
        }
        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
        if (n == 0) break;
    }
}
```

Inicialização

Pi "correto"

Inicia MPI

Obtém #processos

para esse job

Obtém identificador

desse processo

No processo 0, pede ao usuário o número de intervalos

Computação

Envia o #intervalos a todos os processos

GPPD



Calculando Pi (2)

```
h = 1.0 / (double) n;
sum = 0.0;
for (i = myid + 1; i <= n; i += numprocs) {
    x = h * ((double)i - 0.5);
    sum += 4.0 / (1.0 + x*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));
MPI_Finalize();
return 0;
}
```

Calcula as áreas para os intervalos desse processo

Soma os resultados

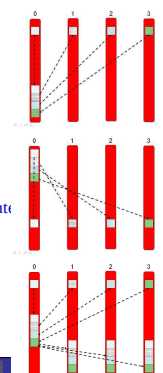
Finalização
No processo 0, imprime o valor estimado para pi e o desvio do valor "correto"

G



Outras comunicações coletivas

- **MPI_Gather**,
 - O processo 'root' agrega buffers de todos os demais processos (n elementos), em um buffer de $p \times n$ elementos.
- **MPI_Scatter**,
 - Um $p \times n$ buffer está "explodido" entre p processos.
- **MPI_Gatherv**, **MPI_Scatterv**
 - Versões generalizadas, com buffers de tamanhos diferentes por processo.
- **MPI_Allgather**
 - = Gather + Bcast.
- **MPI_Alltoall**
 - O nome diz tudo...



nicolas@inf.ufrgs.br

GPPD

Datatypes de usuário

- Os MPI Datatypes básicos para as comunicações são **MPI_INT**, **MPI_FLOAT**, **MPI_CHAR**, **MPI_DOUBLE**...
- Como **comunicar tipos de dados abstratos**?
 - Pode-se usar o "método naïve"
 - Ou:
 - Descreve ao MPI sua estrutura de dados,
 - Efetua um "commit" para criar um novo MPI Data_type personalizado,
 - Usa o Data_type como se fosse um Datatype MPI básico, nas mensagens.

nicolas@inf.ufrgs.br

GPPD

O método naïve (e menos naïve)

- **Método naïve**:
 - memcpy() toda a estrutura de dados num buffer de Bytes, recupera seu tamanho e manda-o como MPI_CHAR.
 - Pouco elegante, e nem um pouco portátil.
- **Solução melhor**: usa o MPI para empacotar os dados:
 - **MPI_Pack**(void* inbuf, int count, MPI_Datatype dtype, void* outbuf, int outcount, int* position, MPI_Comm comm)
 - Empacota os dados em 'inbuf' (= 'count' itens de tipo básico 'dtype'), retorna 'outbuf' e atualiza 'position' para outras chamadas a MPI_Pack().
- **Manda o buffer com tipo básico MPI_PACK**:
 - MPI_Send(outbuf, MPI_Pack_size(outbuf), MPI_PACK, ...)
- **Usa MPI_Unpack(...)** depois do MPI_Recv para desempacotar o buffer.

nicolas@inf.ufrgs.br

GPPD

MPI-2

- "Nova" norma MPI, que prevê novas funcionalidades.
 - Definida em 1998!
 - Apesar de LAM-MPI ter fornecido uma implementação de MPI-2 já em 2000, foi apenas a partir de 2005 que todas as distribuições passaram a implementar MPI-2.
- MPI-2 acrescenta MPI-1.2 com:
 - **Criação dinâmica** de processos,
 - Facilidades para comunicação entre esses processos;
 - **I/O Paralelas**;
 - Remote-Memory Access.

nicolas@inf.ufrgs.br

GPPD

Criação dinâmica de Processos

- MPI-2 possibilita a criação de processos (MPI) processes durante a execução de um programa MPI.
 - Pode-se "spawnar" apenas programas MPI válidos.
 - Pode-se depois mandar esses processos se comunicar.
- **MPI_Comm_spawn**(char* command, char* args[], int maxprocs, MPI_Info info, int root, MPI_Comm com, MPI_Comm* inter_comm, int errors[]);
 - dispara 'maxprocs' novos processos para executar 'command' (= um outro programa MPI).
 - Os 'args' são repassados ao novo programa MPI, como se ele fosse "mpiexec"-utado.
 - Em saída, Spawn retorna um 'inter_comm': um novo comunicador para interconectar os dois conjuntos de processos.
- O tipo abstrato MPI_Comm foi refinado, em MPI-2, para distinguir intras. Inter-communicators.

nicolas@inf.ufrgs.br

GPPD

Comunicações entre os processos criados

- O mecanismo básico continua sendo a troca de mensagens.
 - O processo pai pode usar o inter_comm retornado por Spawn para "conversar" com os processos filhos.
 - Os processos filhos podem usar MPI_Get_comm para determinar o comunicador de seus pais (se têm pais).
- Outra opção é usar **MPI_Intercomm_merge** para fundar o inter_comm com o MPI_COMM_WORLD pre-existente (do lado do pai) e o MPI_COMM_WORLD dos filhos recém-criados.

nicolas@inf.ufrgs.br

GPPD

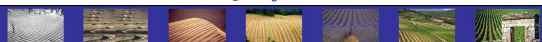
MPI-I/O

- Problema clássico em processamento paralelo: como garantir, no mesmo momento::
 - Acessos paralelos ao sistema de arquivos,
 - Garantir a exclusão mútua necessária?
- Soluções clássicas em MPI:
 - Um processo 'root' centraliza os I/O.
 - Ou cada processo escreve em um arquivo diferente (/tmp).
- Soluções em MPI-2: prover uma nova abstração de arquivos: **MPI_File**.
 - MPI_File_open — MPI_File_write MPI_File_read — MPI_File_close
- As operações num MPI_File são coletivas (conforme definido pelo communicator).
- Usa **MPI_File_view** para especificar quais partes de um MPI_File cada processo irá acessar:


```
MPI_File_set_view(file, rank*BUFSIZE*sizeof(char), MPI_CHAR, ...)
```

nicolas@inf.ufrgs.br

GPPD



Além de MPI...

- MPI é o padrão de fato.
- MPI++ - melhorias dentro do MPI (pesquisa atual)
 - Tolerância a falhas
 - Checkpoint/restarts,
 - É possível no nível do MPI?
 - É eficiente no nível do MPI?
 - Migração de processos.
 - Suporte a arquiteturas heterogêneas
 - MPI-G2
- Entretanto, as limitações do MPI são, na verdade, devidas ao seu modelo de programação.
 - “MPI is the assembly language of parallel programming” (B. Groppe)
 - Talvez não seja Assembly, mas é no máximo C/Fortran.
 - E OO, Templates, Componentes, usar melhor o Compilador?

nicolas@inf.ufrgs.br

GPPD

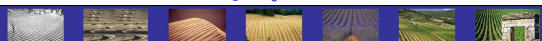


Conclusão do estado da Arte

- **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.

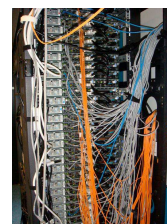
nicolas@inf.ufrgs.br

GPPD



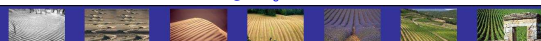
Conclusões

- 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?



nicolas@inf.ufrgs.br

GPPD



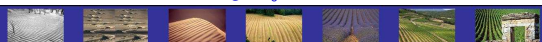
Conclusões (2)

- 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.



nicolas@inf.ufrgs.br

GPPD



Boa sorte!

Nicolas Maillard



nicolas@inf.ufrgs.br

