Semana sobre Programação Massivamente Paralela 2017

MC-MP09. Adaptive MPI (AMPI)

Laércio Pilla (UFSC)

Celso Mendes (INPE)

Esteban Meneses (ITCR)

cc BY SA

Agenda

1. AMPI

a) Introdução a AMPI

b) Transformação de MPI para AMPI

c) Remoção de variáveis globais

d) Balanceamento de carga

e) Checkpointing

f) Visualização de desempenho

Introdução a AMPI

- **Cenário Atual**
  - MPI é o paradigma de programação paralela mais popular
  - Implementações típicas de MPI
    - Não são adequadas para aplicações dinâmicas
      - Variações dinâmicas: movimentação de carga, refinamentos adaptativos, etc.
  - Conjunto de processadores disponíveis
    - Pode não ser o melhor para a expressão natural de algoritmos

CC BY SA

Introdução a AMPI

- **AMPI: Adaptive MPI** (MPI Adaptativo)
  - MPI com virtualização
    - **VPs = *Virtual Processors***
  - Implementação de MPI baseada em Charm++
    - Beneficia-se do sistema de execução do Charm++

Introdução a AMPI

- **AMPI e aderência ao padrão MPI**
  - Aderência quase total ao padrão MPI-1.1
    - Funcionalidades não disponíveis: tratamento de erros, interface de *profiling*
  - Aderência quase total ao padrão MPI-2.2
    - Comunicação unilateral
    - ROMIO integrado para I/O paralelo
    - Funcionalidades não disponíveis: gerenciamento dinâmico de tarefas, falta de suporte mais completo em algumas linguagens

Introdução a AMPI

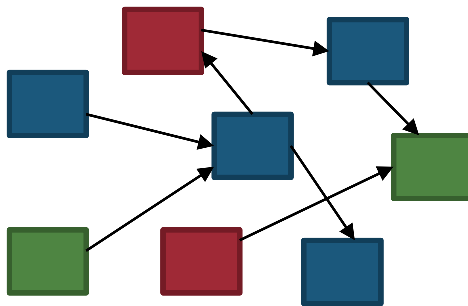
- **AMPI e aderência ao padrão MPI**
  - Aderência parcial ao padrão MPI-3
    - Operações coletivas sem bloqueio (algumas já existiam como extensões em AMPI)
    - Operações coletivas em vizinhança
    - Funcionalidades não disponíveis: interface de ferramentas, otimizações com janelas, e outras
  - Extensões de AMPI ao padrão MPI
    - Funções definidas como AMPI_xxx
    - Diversas funcionalidades – algumas são mostradas nos próximos slides!

cc BY SA

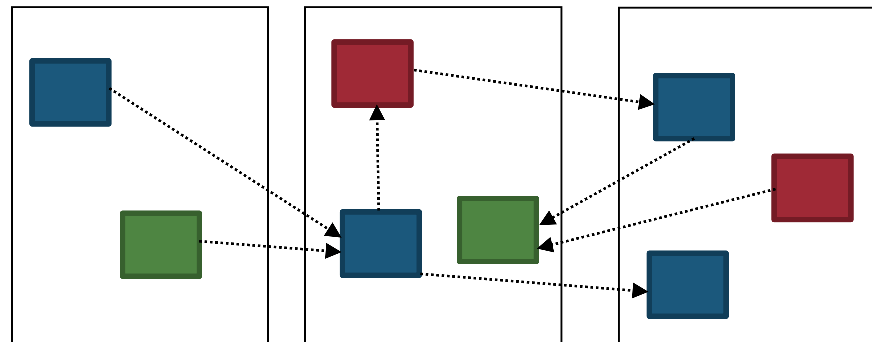
Introdução a AMPI

- **Ideia básica da virtualização**
  - Cada objeto é visto como um VP
  - A pessoa programadora especifica as interações entre objetos (VPs)

Visão de Programação



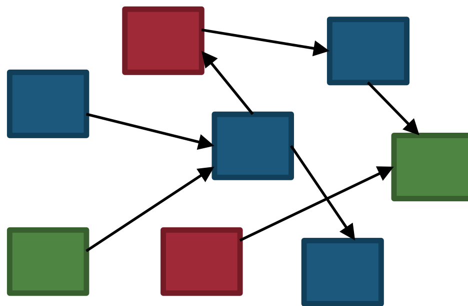
Implementação no sistema



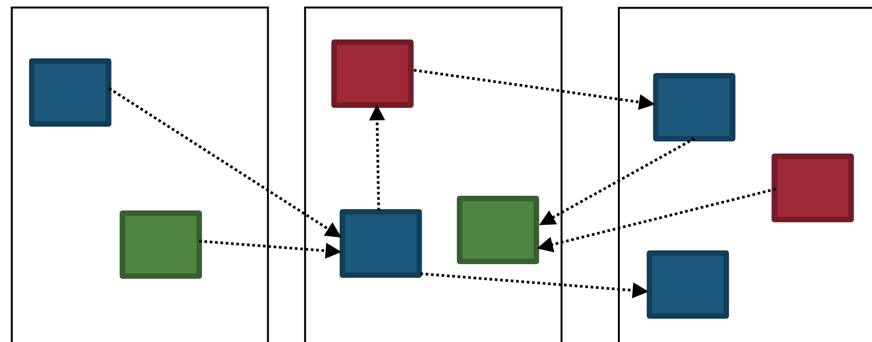
Introdução a AMPI

- **Ideia básica da virtualização**
  - O sistema de execução mapeia os VPs para os processadores
  - Tipicamente, **#VPs >> #processadores**

Visão de Programação



Implementação no sistema

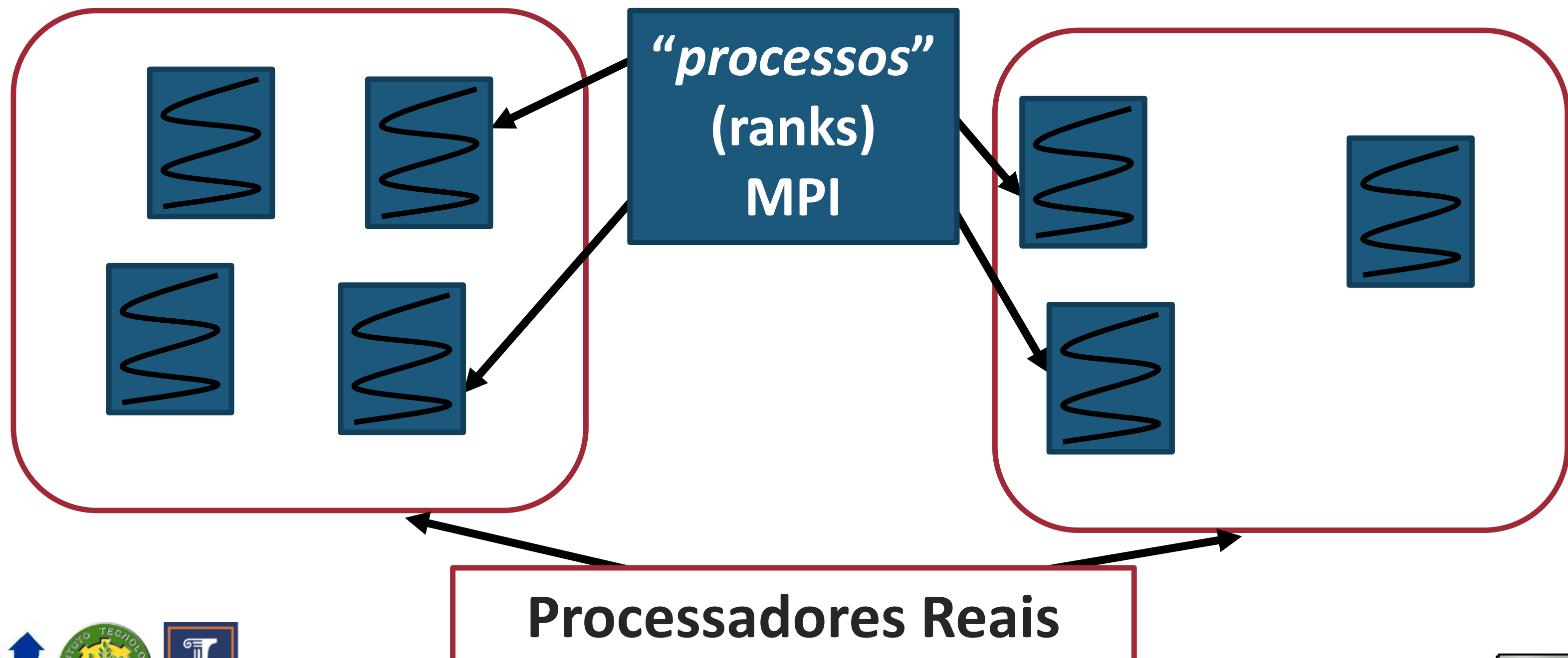


Introdução a AMPI

- **Atributos de Charm++ explorados por AMPI**
  - *Threads* em nível de usuário
    - Sem bloqueio da CPU
    - Leves: mudança de contexto é rápida (~ 1μs)
    - *Threads* podem migrar entre processadores dinamicamente

Introdução a AMPI

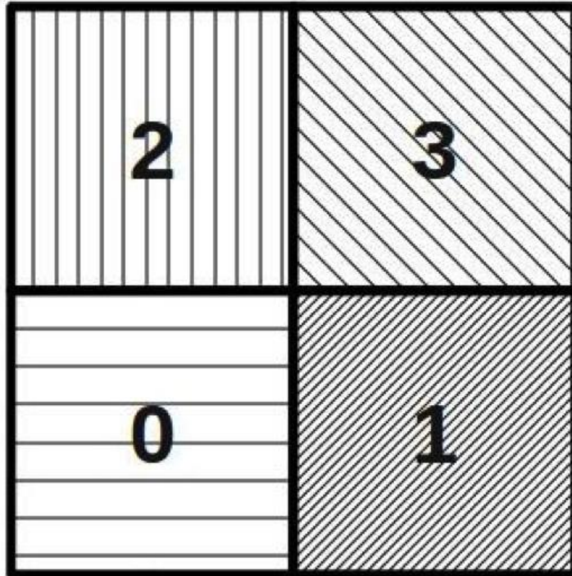
- De MPI para AMPI
  - Ranks de MPI são transformados em VPs
  - Cada VP é uma *thread* de usuário embutida em um objeto de Charm++



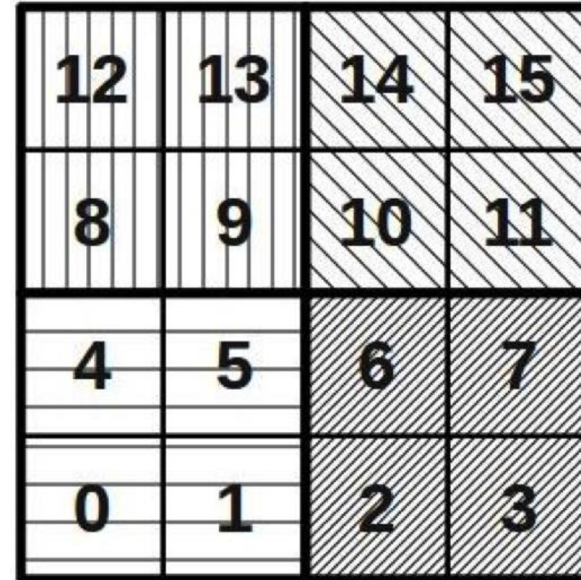
cc BY SA

Introdução a AMPI

MPI: P=4, ranks=4



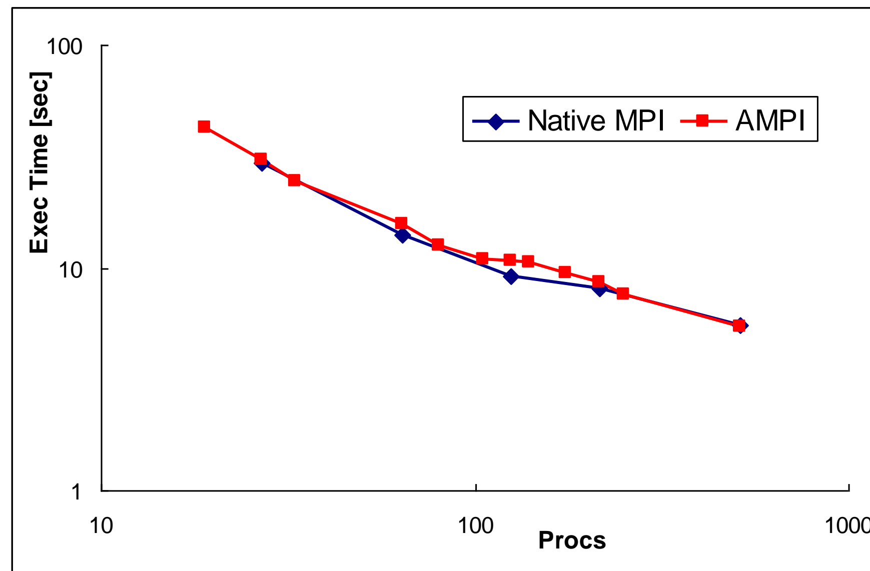
AMPI: P=4, VP=ranks=16



- **Vantagens da virtualização de AMPI**
  - Domínios menores podem usar melhor caches
  - Quando uma *thread* bloqueia, outra pode rodar

Introdução a AMPI

- **Flexibilidade**
  - Execuções com quaisquer #processadores
  - Adequa-se sempre à natureza dos algoritmos



Cenário: cálculo de stencil 3D com tamanho 240^3 no sistema Lemieux.

AMPI roda com qualquer número de processadores (ex: 19, 33, 105), enquanto MPI Nativo precisa de $P=K^3$

cc BY SA

Introdução a AMPI

- **Coletivas assíncronas**

  Existem em AMPI mesmo antes de MPI-3

  Idéia básica: execução assíncrona

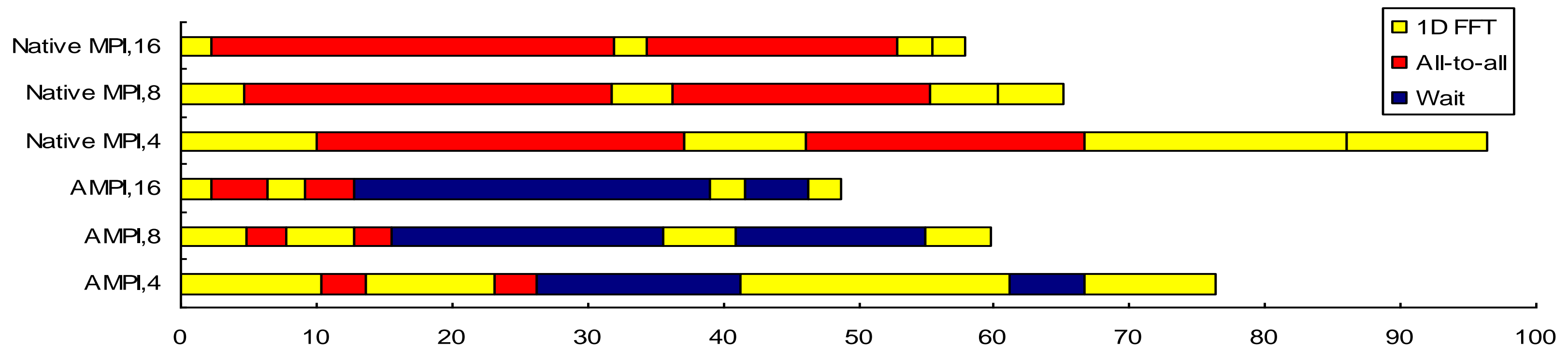
  - Operação coletiva é chamada, sem bloqueio
  - Teste/espera por sua conclusão
  - Enquanto isso, outras computações podem prosseguir e utilizer a CPU

```
MPI_Ialltoall( … , &req);
/* outras computações */
MPI_Wait(req);
```


UFSC INPE CC BY SA

Introdução a AMPI

- **Coletivas assíncronas (2)**



Componentes de tempo (ms) num benchmark de FFT-2D

- VPs implementados como *threads*
- Sobreposição de computação com tempo de espera das operações coletivas
- Tempo total de execução é reduzido, em todos os casos!

Introdução a AMPI

- Exemplo: **BRAMS**
  - Execução em 64 processadores - Kraken (ORNL)

Configuração	Tempo de Execução (s)
64 AMPI threads (sem virtualização)	4970
256 AMPI threads	3857
1024 AMPI threads	3713
2048 AMPI threads	4437

  - Dois tipos de ganho de desempenho
    - Sobreposição de computação e comunicação
    - Melhor utilização de cache
      - Ref.: *Rodrigues et al, HiPC 2010*

cc BY SA

Introdução a AMPI

- **Interoperabilidade com Charm++ e MPI**
  - Charm++ tem várias bibliotecas de suporte
    - É possível usar tais bibliotecas rodando código de Charm++ num programa AMPI
    - Similarmente, é possível rodar códigos MPI em programas Charm++
  - Também é possível juntar vários códigos MPI numa única aplicação AMPI
    - Ex: um código MPI para fluidos, outro para sólidos
    - É preciso escrever uma rotina de nível mais alto para criar e coordenar os vários threads AMPI

cc BY SA

Agenda

1. AMPI

a) ~~Introdução a AMPI~~

b) Transformação de MPI para AMPI

c) Remoção de variáveis globais

d) Balanceamento de carga

e) Checkpointing

f) Visualização de desempenho

cc BY SA

Transformação de MPI para AMPI

- **Como escrever um programa com AMPI**
  - Escreva seu programa normal MPI e
    - Compile e execute com Charm++
      - Monte o Charm++ com target=AMPI
    - Compile e ligue com charmc ou ampicc
    - Inclua charm/bin/ no caminho de busca

      > *charmc -o hello hello.c -language ampi* ou

      > *ampicc -o hello hello.c*
    - Execute com charmrun

      > *./charmrun hello*

cc BY SA

Transformação de MPI para AMPI

- **Como usar AMPI no Santos Dumont**
  - Escreva seu programa normal MPI e
    - Obtenha e construa o AMPI no S.Dumont
      - *git clone https://charm.cs.illinois.edu/gerrit/charm*
      - *cd charm*
      - *./build AMPI mpi-linux-x86_64*
    - Compile incluindo charm/mpi-linux-x86_64/bin/ no caminho de busca

      > *ampicc -o hello hello.c*
    - Execute em um script apropriado, com srun:

      > *srun hello*

cc BY SA

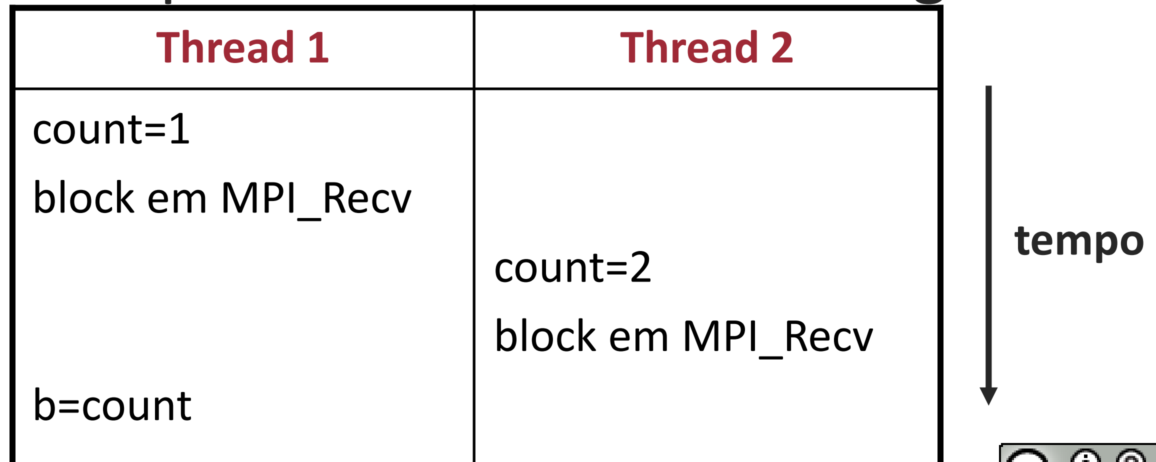
Transformação de MPI para AMPI

- **Como escrever um programa com AMPI**
  - É possível rodar muitos programas MPI com Charm++/AMPI
    - Há exceções (a serem discutidas)
  - *mpirun –npK prog = charmrun prog +pK*
  - *machinefile* em MPI = *nodelist* em Charm++
  - [S.Dumont] *srun –n prog*
  - Exemplo: hello.c

CC BY SA

Transformação de MPI para AMPI

- **Uso de variáveis globais**
  - Deve ser sempre evitado
  - Perigosas em ambientes de múltiplas threads
    - Podem ser modificadas por qualquer thread
    - Similar a variáveis globais em OpenMP
  - Exemplo: assuma que *count* é uma variável global



Thread 1	Thread 2
count=1	
block em MPI_Recv	
	count=2
	block em MPI_Recv
b=count	

Valor incorreto (2) é lido!

Transformação de MPI para AMPI

- **Execução com AMPI**
  - Mesmo em um único processador, é possível rodar com várias threads
  - Execução com múltiplos VPs:
    - Opção **+p** : define #processadores físicos
    - Opção **+vp** : define #VPs
  - Exemplo:
    - > *./charmrun jacobi +p3 +vp16*
    - > [S.Dumont] *srun –n 3 jacobi +vp 16*
      - 16 ranks em 3 processadores
  - Exemplo: jacobi.c

cc BY SA

Transformação de MPI para AMPI

- **Execução com AMPI**
  - Múltiplos mapeamentos iniciais de VPs para processadores disponíveis
    - > *charmrun hello +p3 +vp6 +mapping <map>*
    - > [S.Dumont] *srun –n 3 hello +vp 6 +mapping <map>*
      - RR_MAP: Round-Robin (cíclico) *{(0,3)(1,4)(2,5)}*
      - BLOCK_MAP: Em blocos (padrão) *{(0,1)(2,3)(4,5)}*
      - PROP_MAP: Proporcional à velocidade dos processadores *{(0,1,2,3)(4)(5)}*
  - Exemplo: mapping.c

cc BY SA

Transformação de MPI para AMPI

- **Execução com AMPI**
  - Especificação do tamanho de pilha por thread
    - Possível definir pilhas menores ou maiores
      - Pilha é exclusiva para cada thread
  - Definição do tamanho de pilha é feita com a opção *+tcharm_stacksize*
    - > ./charmrun hello +p2 +vp8 +tcharm_stacksize 8000000
    - > [S.Dumont] *srun –n 2 hello +vp8 +tcharm_stacksize 8000000*
      - Tamanho de pilha default: 1 MB por thread
  - Exemplo: bigstack.c

cc BY SA

Transformação de MPI para AMPI

- **Fortran**
  - Ponto de entrada “MPI_Main” precisa ser mudado manualmente

```
program pgm    →   subroutine MPI_Main
...                  ...
end program        end subroutine
```

- Em C, ponto de entrada “main()” é mudado automaticamente ao compilar via mpi.h
  - *mpi.h* deve ser incluído pelo arquivo contendo a declaração de *main()*

cc BY SA

Agenda

1. AMPI

a) ~~Introdução a AMPI~~

b) ~~Transformação de MPI para AMPI~~

c) **Remoção de variáveis globais**

d) **Balanceamento de carga**

e) **Checkpointing**

f) **Visualização de desempenho**

Remoção de variáveis globais

- **Remover variáveis globais, quando possível**
  - Se não for possível remover, ***privatizar*** as variáveis globais
    - Empacotá-las numa struct/TYPE ou class
    - Alocar struct/type no heap ou stack

Código Original

```
MODULE shareddata
  INTEGER :: myrank
  DOUBLE PRECISION :: xyz(100)
END MODULE
```

Código AMPI

```
MODULE shareddata
  TYPE chunk
    INTEGER :: myrank
    DOUBLE PRECISION :: xyz(100)
  END TYPE
END MODULE
```

Remoção de variáveis globais

Código Original

```
PROGRAM MAIN
  USE shareddata
  include 'mpif.h'
  INTEGER :: i, ierr
  CALL MPI_Init(ierr)
  CALL MPI_Comm_rank(
       MPI_COMM_WORLD,
       myrank, ierr)
  DO i = 1, 100
    xyz(i) = i + myrank
  END DO
  CALL subA
  CALL MPI_Finalize(ierr)
END PROGRAM
```

Código AMPI

```
SUBROUTINE MPI_Main
  USE shareddata
  USE AMPI
  INTEGER :: i, ierr
  TYPE(chunk), pointer :: c
  CALL MPI_Init(ierr)
  ALLOCATE(c)
  CALL MPI_Comm_rank(
       MPI_COMM_WORLD,
       c%myrank, ierr)
  DO i = 1, 100
    c%xyz(i) = i + c%myrank
  END DO
  CALL subA(c)
  CALL MPI_Finalize(ierr)
END SUBROUTINE
```


cc BY SA

Remoção de variáveis globais

Código Original

```
SUBROUTINE subA
  USE shareddata
  INTEGER :: i
  DO i = 1, 100
    xyz(i) = xyz(i) + 1.0
  END DO
END SUBROUTINE
```

Código AMPI

```
SUBROUTINE subA(c)
  USE shareddata
  TYPE(chunk), pointer :: c
  INTEGER :: i
  DO i = 1, 100
    c%xyz(i) = c%xyz(i) + 1.0
  END DO
END SUBROUTINE
```

- Substituição das variáveis globais por argumentos de funções
  - Variáveis estáticas têm o mesmo problema
  - C: struct + * + malloc + free

cc BY SA

Remoção de variáveis globais

- **Formato ELF e variáveis globais/estáticas**
  - Variáveis globais/estáticas não são thread-safe
    - Ideia: chavear as variáveis globais ao mudar contexto
  - Padrão *Executable and Linking Format* (ELF)
    - Executável tem uma *Global Offset Table* (GOT) contendo informação sobre todas as variáveis globais
    - Ponteiro para a GOT é armazenado no registrador %ebx

Remoção de variáveis globais

- **Formato ELF e variáveis globais/estáticas**
  - Padrão *Executable and Linking Format* (ELF)
    - Basta mudar o ponteiro para a GOT ao mudar o contexto entre threads
    - Na prática, cada thread passa a ter sua própria versão da GOT
  - Suportado em Linux, Solaris 2.x, e outros
  - Disponível em Charm++/AMPI
    - Invocado pela opção de compilação ***-swapglobals***
  - Exemplo: globals.c

Remoção de variáveis globais

- **Privatização via TLS (*Thread Local Store*)**
  - Padrão implementado por vários compiladores modernos
    - Variáveis críticas são “anotadas”: ***__thread int var;***
  - Requer coordenação entre o compilador e o sistema de execução para poder ser usado em programas AMPI
    - Compilador Fortran: gfortran modificado (trata globais/estáticas via TLS)
      - Modificação feita por Eduardo Rodrigues

Remoção de variáveis globais

- **Privatização via TLS (*Thread Local Store*)**
  - Suporte do Charm++
    - Montagem com flag ***-tlsglobals***
  - Vantagens de TLS sobre swapglobals:
    - Evita cópia de variáveis durante a mudança de contexto
    - Tempo da mudança de contexto torna-se independente do número de variáveis globais/estáticas no código

CC BY SA

Remoção de variáveis globais

- **Outras ferramentas para privatização**
  - **Pacote Photran**
    - Desenvolvido em Illinois pelo grupo do Prof. Ralph Johnson
    - Baseado em técnicas de refatoração de código
    - Funciona apenas com códigos em Fortran

CC BY SA

Remoção de variáveis globais

- **Outras ferramentas para privatização**
  - **Pacote Rose**
    - Desenvolvido no Livermore Lab. pelo Dr. Dan Quinlan
    - Nova ferramenta de reestruturação de código desenvolvida no PPL, utilizando Rose
    - Funciona com aplicações em C/C++ e Fortran
  - Ambas as ferramentas exigem contato específico (suporte pode variar)

Agenda

1. AMPI

a) ~~Introdução a AMPI~~

b) ~~Transformação de MPI para AMPI~~

c) ~~Remoção de variáveis globais~~

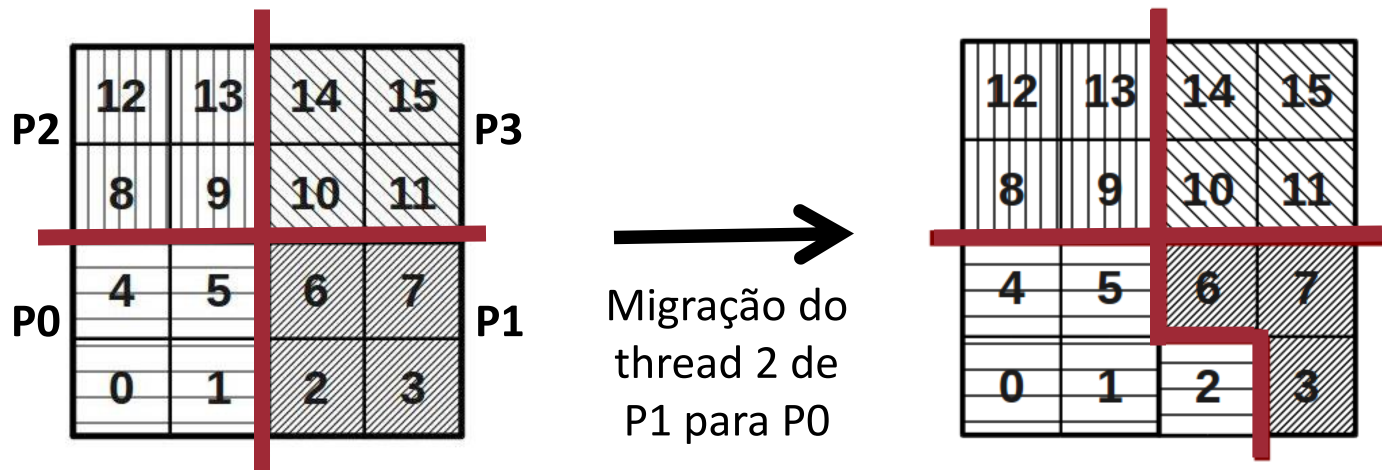
d) **Balanceamento de carga**

e) **Checkpointing**

f) **Visualização de desempenho**

Balanceamento de carga

- **Movimentação de threads AMPI**
  - Possibilidade de melhorar o balanceamento de carga entre os processadores
  - Potencial para evitar falhas na execução devido a problemas em um processador específico



cc BY SA

Balanceamento de carga

- **Dificuldade para migrar threads**
  - Todos os dados da thread devem ser enviados ao novo processador e permanecer válidos (inclusive ponteiros)
- Dois mecanismos para transferência de dados

a) Empacotar/desempacotar os dados

- Necessita registrar variáveis que vão ser movidas
- Dados são automaticamente empacotados na origem e desempacotados no destino final

cc BY SA

Balanceamento de carga

- **Pack / Unpack (PUP)**

AMPI

```
struct meusdados {
        int i, t;
        char nome[10];
        int *valores;
}
```

```
void meusdadospup ( pup_er p, struct
meus dados c ) {
        pup_int( p, c.i );
        pup_int( p, c.t );
        pup_chars ( p, c.nome, 10 );
        pup_doubles ( p, c.valores, c.t );
}
```

Charm++

```
class minhaClasse {
        int i, t;
        char nome[10];
        int *valores;
}
```

```
void minhaClasse::pup ( PUP::er &p ) {
        p|i;
        p|t;
        PUParray ( p, nome, 10 );
        PUParray ( p, valores, t );
}
```


CC BY SA

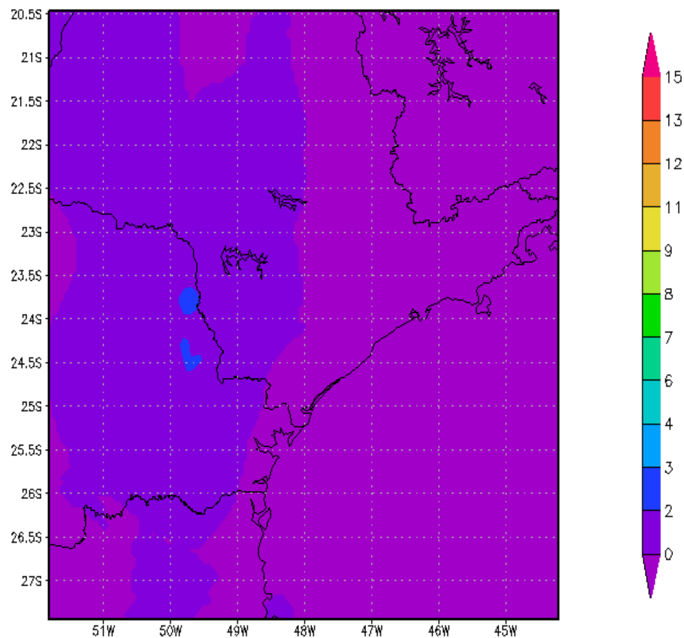
Balanceamento de carga

- Dois mecanismos para transferência de dados
  b) isomalloc
  - Não requer trabalho nenhum de programação
  - Memória é alocada dentro de fatias exclusivas de cada thread
  - Numa migração, toda a fatia do thread é transferida
  - Selecionado com o flag de montagem
    - ***-memory isomalloc***

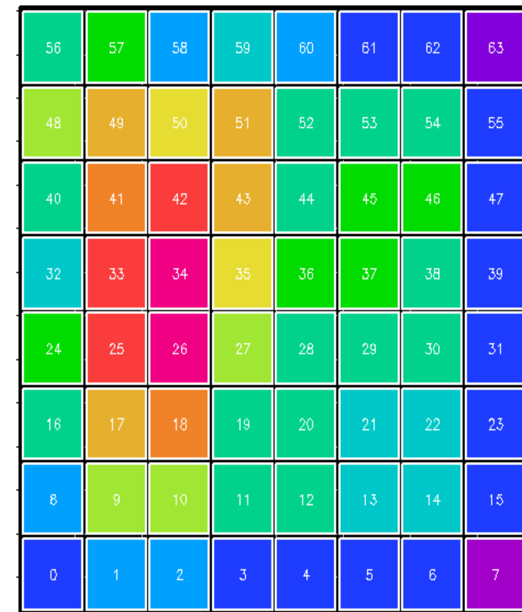
Balanceamento de carga

- **Desbalanceamento de carga**
  - Prejudicial para aplicações dinâmicas
  - Exemplo: BRAMS

Previsão, Fev.2010



Carga dos Processadores (P=64)



Balanceamento de carga

- **Framework de balanceamento de carga**
  - **Arcabouço externo a aplicações** para o balanceamento de carga
    - **Definição de algoritmos para o remapeamento de VPs (ou chares) dado um objetivo**
      - Melhorar comunicação
      - Reduzir o consumo de energia ou temperatura de núcleos
      - Equilibrar o tempo de execução nos vários núcleos

Balanceamento de carga

- **Princípio da persistência**
  - A carga computacional e padrões de comunicação de aplicações científicas tendem a persistir ao longo do tempo
  - **O passado recente é uma boa estimativa do futuro próximo**
    - Informações das aplicações são coletadas e usadas para a tomada de decisões de escalonamento global

cc BY SA

Balanceamento de carga

- **Balanceamento de carga automático**
  - **AMPI: AMPI_Migrate(hints)**
    - Charm++: AtSync() + ResumeFromSync()
  - Chamada coletiva informando ao sistema de execução que a thread está pronta para ser migrada, se for necessário
    - Se o sistema de execução julgar que o desbalanceamento não é muito alto, não há migrações

cc BY SA

Balanceamento de carga

- **Balanceamento de carga automático**
  - **Quando informações de carga são obtidas?**
    - Funções para ligar e desligar a medição de carga
      - AMPI_Load_start_measure(), AMPI_Load_stop_measure()
    - Por default, medições são sempre coletadas
    - Função para associar um certo nível de carga a um thread
      - AMPI_Load_set_value()

cc BY SA

Balanceamento de carga

- **Balanceamento de carga automático**
  - Caso houver migrações, elas consistem em
    - Avaliação da quantidade de dados e empacotamento destes no processador de origem
    - Envio de dados da pilha e dados empacotados para o destino
    - Desempacotamento dos dados no processador de destino
    - Reinício da execução em cada processador

Balanceamento de carga

- **Uso de balanceadores**
  - Inserção de chamada no código-fonte da aplicação
  - Link deve ser feito com módulos LB
    - > ***ampicc –o pgm hello.o -module CommonLBs***
  - Executar com a opção *+balancer*
    - > ***./charmrun pgm +p4 +vp16 +balancer GreedyLB***
    - > **[S.Dumont]** ***srun –n4 pgm +vp16 +balancer GreedyLB***
  - Exemplo: loadbal.c

cc BY SA

Balanceamento de carga

- **Uso de balanceadores**
  - Vários balanceadores disponíveis c/ Charm++
    - Cada balanceador segue uma certa estratégia
      - **GreedyLB**: envia objetos mais custosos aos processadores menos carregados
      - **GreedyCommLB**: estende GreedyLB considerando o grafo de comunicação entre processadores
      - **RefineLB**: Limita o número de threads migrados
      - **Outros**: várias outras políticas
    - Novos balanceadores podem ser desenvolvidos

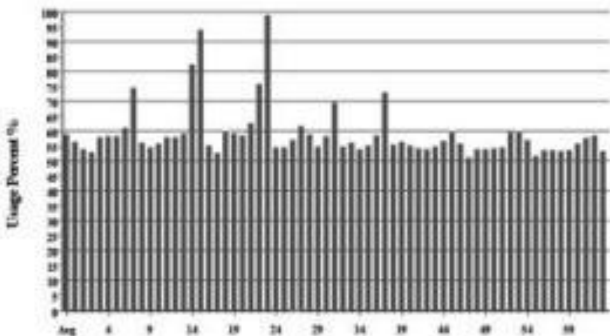
cc BY SA

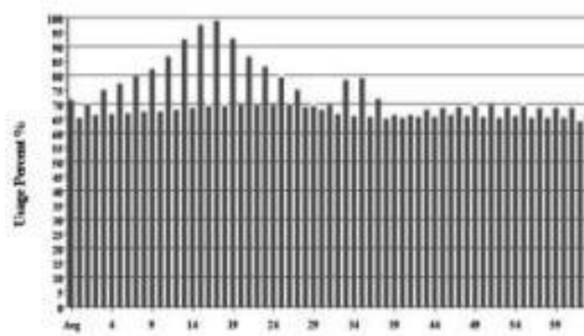
Balanceamento de carga

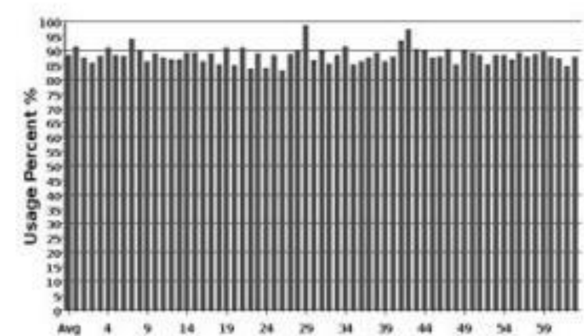
- **Ganhos obtidos para o BRAMS**

Configuração	Tempo de Execução (s)
64 AMPI threads (sem virtualização)	4970
1024 AMPI threads (só virtualização)	3713 (-25%)
1024 AMPI threads + Balanceamento	3367 (-32%)

Utilização dos 64 processadores em cada configuração:







Balanceamento de carga

- **Ganhos obtidos para Ondes3D**
  - SD = sobredecomposto (512 VPs/ 32 núcleos)

Desbalanceamento

Dinamicidade

Tempo de iterações (s)

5
4
3
2
1
0

20 60 100 140 180 220 260 300 340 380 420 460 500 540 580

Iterações

Tempo base — Carga média — Tempo base SD — Carga média SD

CC BY SA

Balanceamento de carga

- **Ganhos obtidos para Ondes3D**
  - SD = sobredecomposto (512 VPs/ 32 núcleos)

Tempo de iterações (s)

5
4
3
2
1
0

20 60 100 140 180 220 260 300 340 380 420 460 500 540 580

Iterações

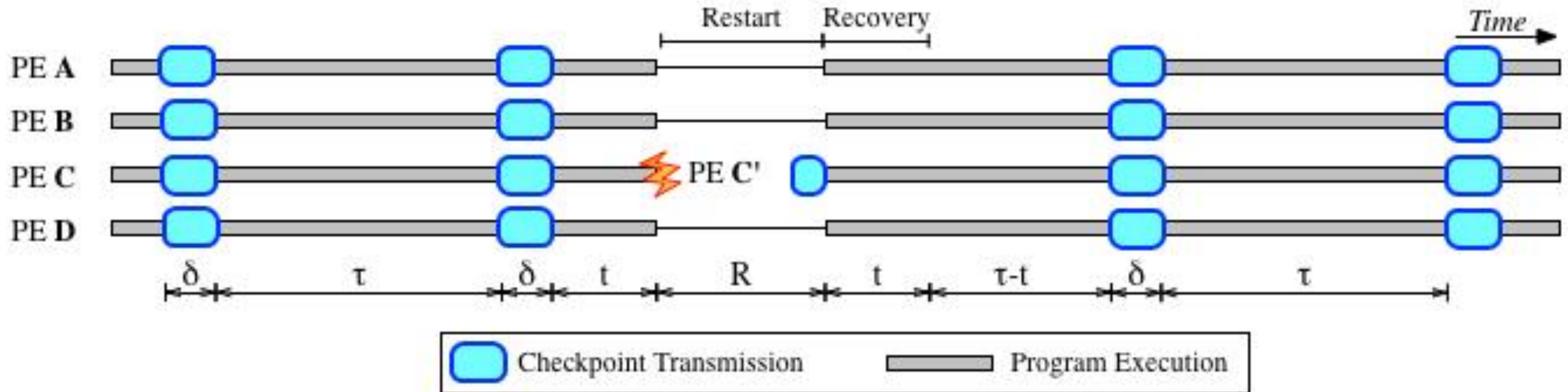
Tempo base — Carga média SD — HwTopoLB — NucoLB

cc BY SA

Agenda

1. **AMPI**
   a) ~~Introdução a AMPI~~
   b) ~~Transformação de MPI para AMPI~~
   c) ~~Remoção de variáveis globais~~
   d) ~~Balanceamento de carga~~
   e) **Checkpointing**
   f) **Visualização de desempenho**

Checkpointing



- Usos
  - **Execução particionada**
  - **Tolerância a faltas** a erros de hardware
  - **Análise** post mortem
  - Exploração de execuções alternativas

cc BY SA

Checkpointing

- **Checkpoint com chamada coletiva**
  - Em disco: ***MPI_Checkpoint(DIRNAME)***
  - Em memória: ***MPI_MemCheckpoint(void)***
  - Checkpoint síncrono
- **Reinício pode ser manual ou automático**
  - Em disco: ***> ./charmrun pgm +p4 +restart DIRNAME***
  - Em memória: detecção automática de falhas, e reinício

cc BY SA

Checkpointing

- **Chamadas em Charm++**
  - Checkpoint baseado em discos

```
void CkStartCheckpoint(char* dirname, const CkCallback& cb);
```

Fonte (.cpp)

```
. . .  CkCallback cb(CkIndex_Hello::SayHi(),helloProxy);
CkStartCheckpoint("log",cb);
```

Agenda

1. AMPI

a) ~~Introdução a AMPI~~

b) ~~Transformação de MPI para AMPI~~

c) ~~Remoção de variáveis globais~~

d) ~~Balanceamento de carga~~

e) ~~Checkpointing~~

f) Visualização de desempenho

CC BY SA

Visualização de desempenho

- **Ferramenta de Charm++: Projections**
  - Instrumentar o programa, visualizar os dados de desempenho
  - Uso
    - Instalação: build AMPI com ***--enable-tracing***
    - Ligação do código do usuário

      > ***ampicc –o prog prog.c –tracemode projections***
        - Captura informações detalhadas durante a execução

      > ***ampicc –o prog prog.c –tracemode summary***
        - Acumula apenas um sumário dos dados coletados

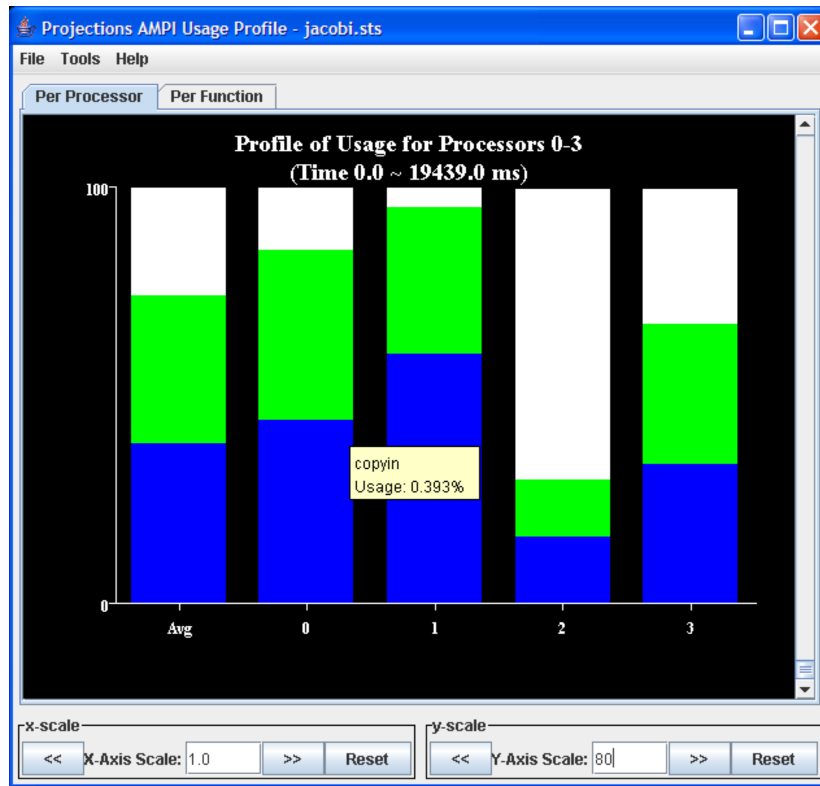
CC BY SA

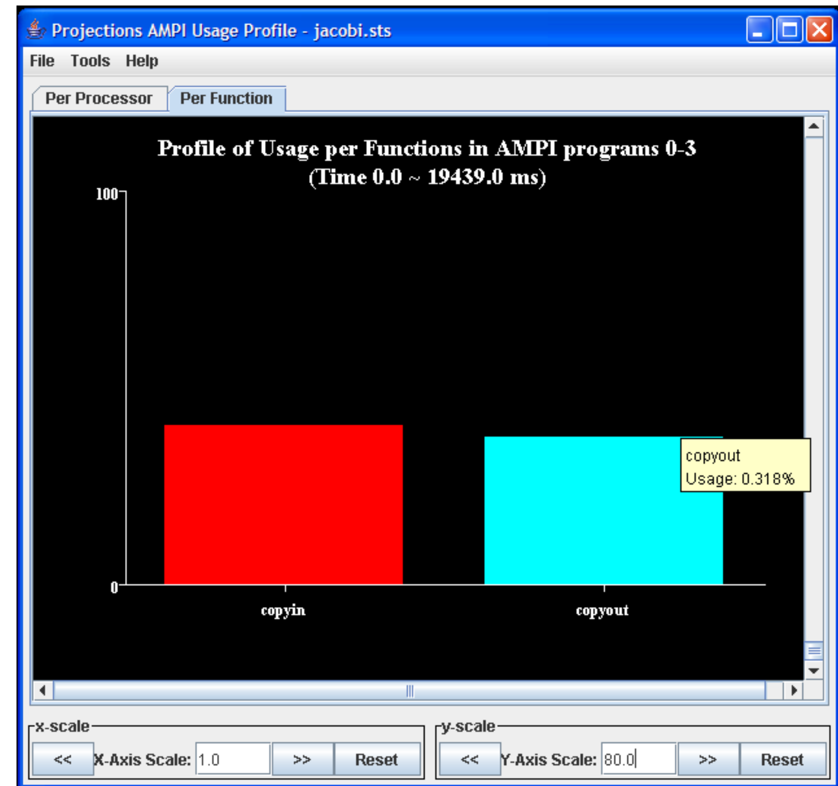
Visualização de desempenho

- **Projections com AMPI**
  - Registrar as chamadas a funções de interesse
    - ***extern int traceRegisterFunction(const char *name, int idx);***
  - Embutir o fragmento de código a ser rastreado
    - ***extern void traceBeginFuncProj(char *name,char *filename, int line);***
    - ***. . . <fragmento de código > . . .***
    - ***extern void traceEndFuncProj(char *name);***

cc BY SA

Visualização de desempenho





AMPI - Sumário

- **Resumo:**
  - AMPI: implementação de MPI via Charm++
    - Ranks de MPI transformados em objetos/threads
    - Mapeamento automático objetos↔processadores
    - Objetos podem migrar dinamicamente
  - Desenvolvimentos correntes
    - Aumento de aderência ao padrão MPI-3
    - Uso de RDMA em suporte a MPI/RMA
    - Otimização da integração AMPI/OpenMP

cc BY SA