スケーラブルシステムズ株式会社

OpenMPプログラミングに関するトピックスの紹介

“Many core”CPU

- 2012(?)に想定される計算ノード 1)
  - Node : 960 GFLOPS/CPU
    - Many core CPU, 48 cores, 2.5GHz,共有キャッシュ
    - シンプルな実行コア(in-orderでSMT機能付き）
    - メモリバンド幅を最大限に活用するアーキテクチャ
    - SIMDベクトルユニット.... 8 FLOP / cycle / core
- 複数の計算ノードがNUMA構成で接続
- 複数の計算サーバ、ブレードでクラスタ構成

➡ 効率の良い並列化が求められる

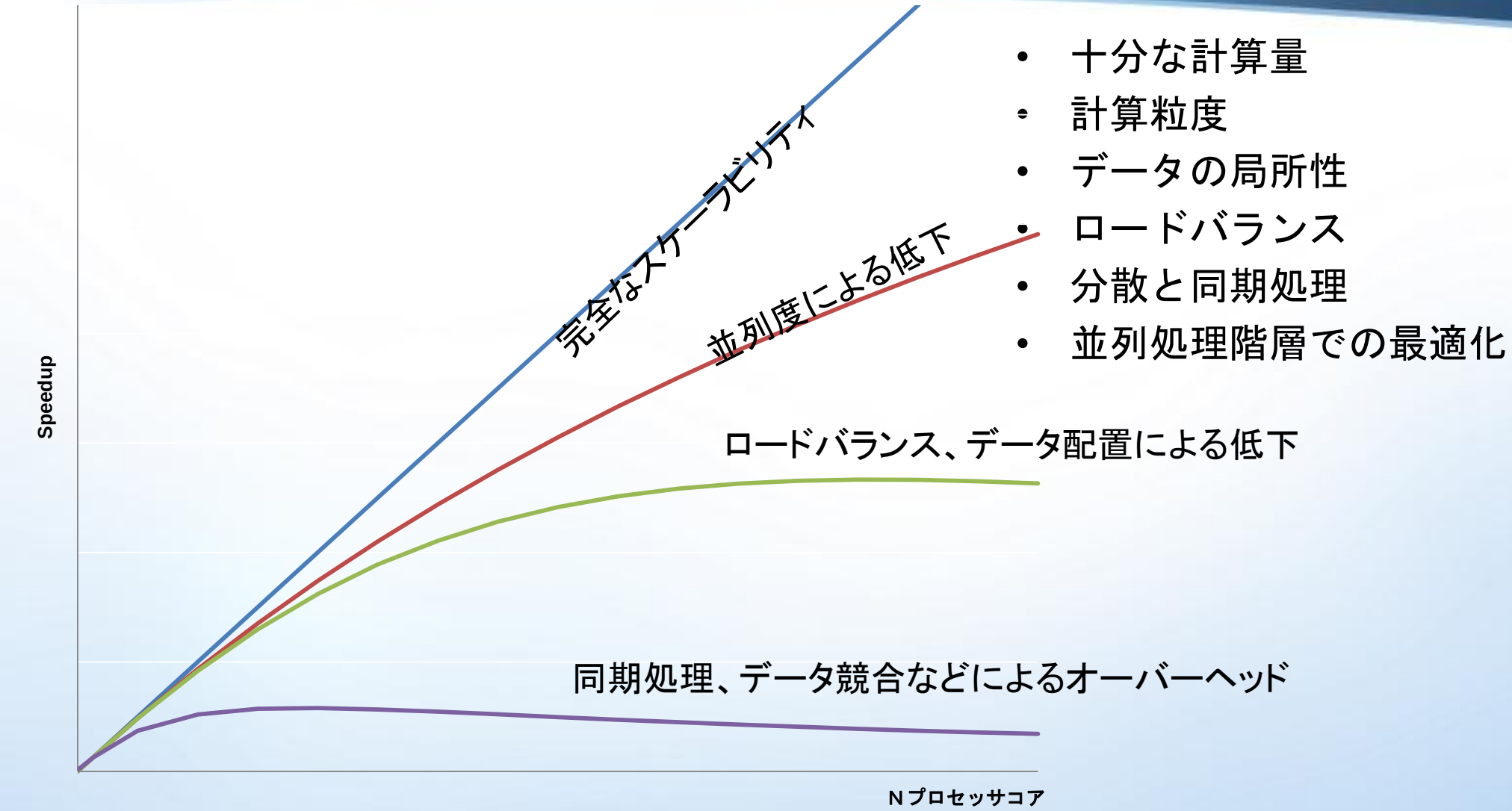
1) 仮定として想定したプロセッサに基づく推察

並列プログラミングで留意点

- 十分な計算量(Amdahl's Law)
- 計算粒度
- データの局所性
- ロードバランス
- 分散と同期処理
- 並列処理階層での最適化

逐次処理(シングルスレッド)アプリケーションと比較しても検討課題が多いことが、並列処理をより困難にしています。

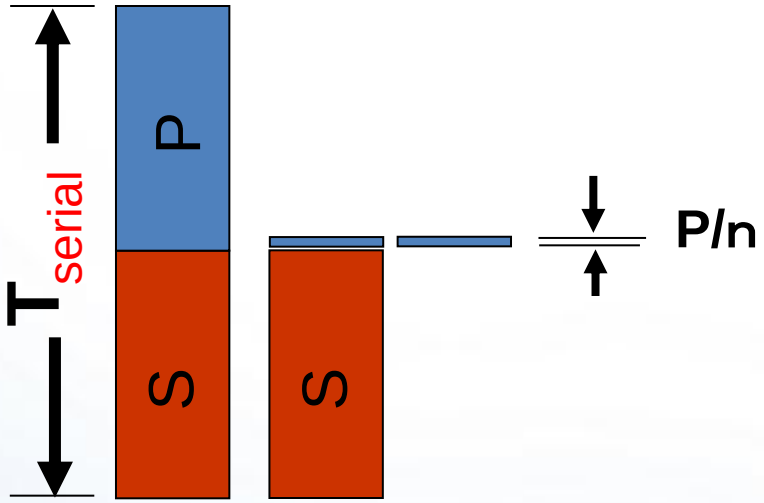
並列プログラミングで留意点



- 十分な計算量
- 計算粒度
- データの局所性
- ロードバランス
- 分散と同期処理
- 並列処理階層での最適化

アムダールの法則

- 並列処理での性能向上の上限値（スケーリング）



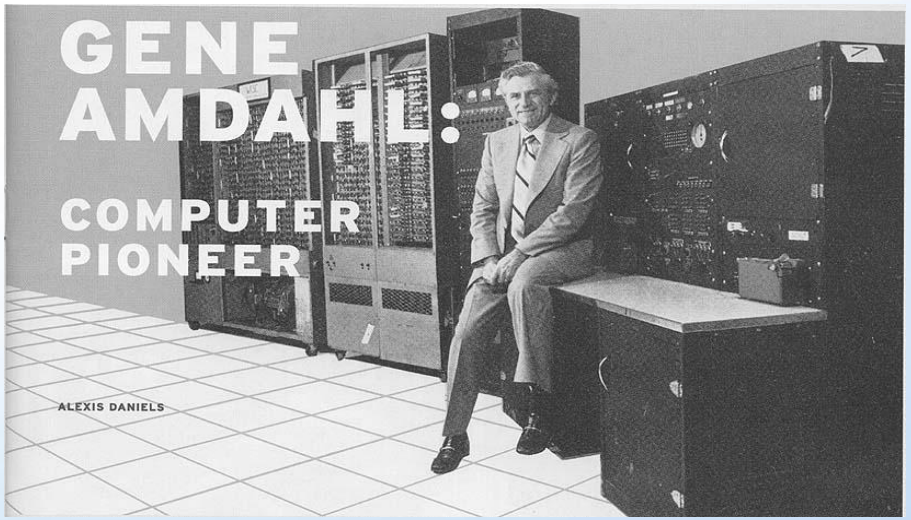
$$T_{parallel} = (S + P/n)\, T_{serial} + O$$

n = number of processors

$$Speedup = T_{serial} / T_{parallel} = 1 / (S + P/n)$$

プログラムの逐次処理部分
（非並列処理）部分の排除が必要

例えば、n=∞, P = 0.5 の場合
Speedup = 1.0 / (0.5 + 0) = 2.0



グスタフソンの法則

●アムダールの法則
作業負荷や問題の規模が一定であることを仮定

●グスタフソンの法則（Gustafson-Barsis‘ law）
並列処理では問題の規模や作業負荷がプロセッサコア数に比例して大きくなり、その負荷増加は逐次処理部分に影響しないことを仮定

「アムダールの法則の限界から並列処理を救い出すことが可能？」

適用出来る問題と利用環境に大きな制限がある

$$T_{parallel} = \{S + P/n\} T_{serial} + O$$

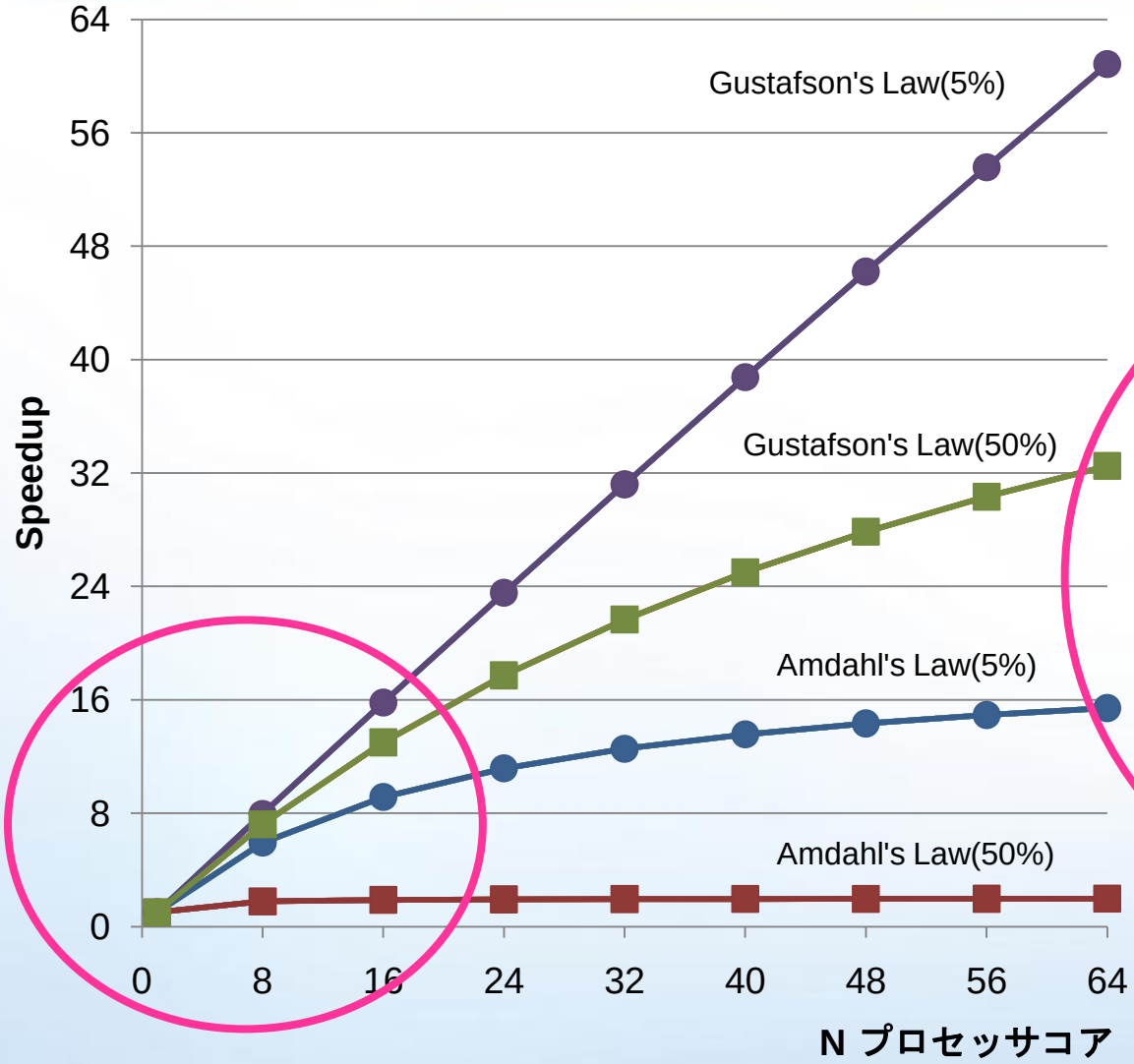
$$Speedup = T_{serial} / T_{parallel} = 1 / (S + P/n)$$

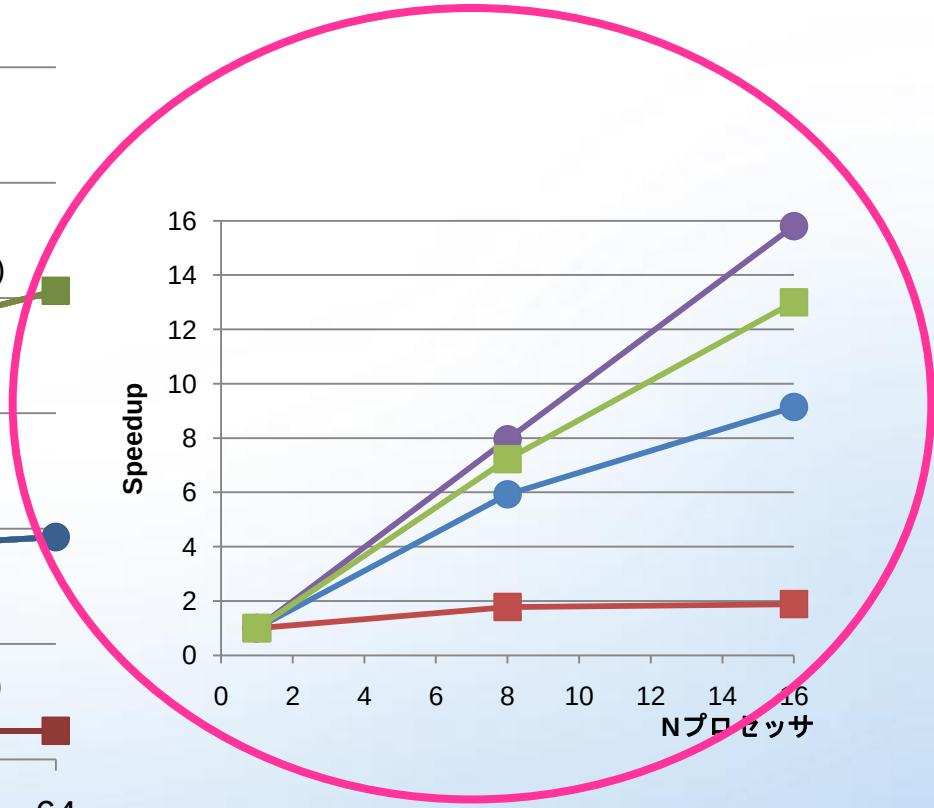
$$T_{serial} = (S + n \times P) T_{parallel}$$

$$Speedup = T_{serial} / T_{parallel} = (S + n \times P)$$

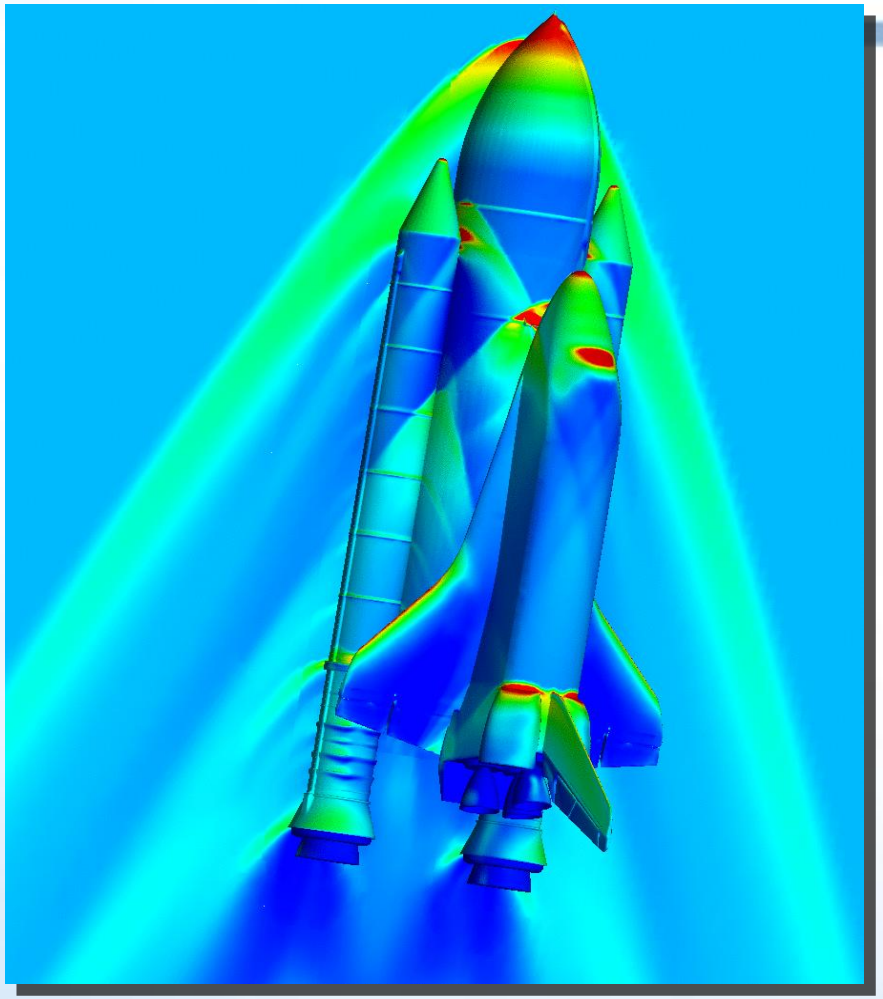
例えば、n=16, P = 0.5 の場合
Speedup = 0.5 + 16 x 0.5 = 8.5

スケーラビリティ

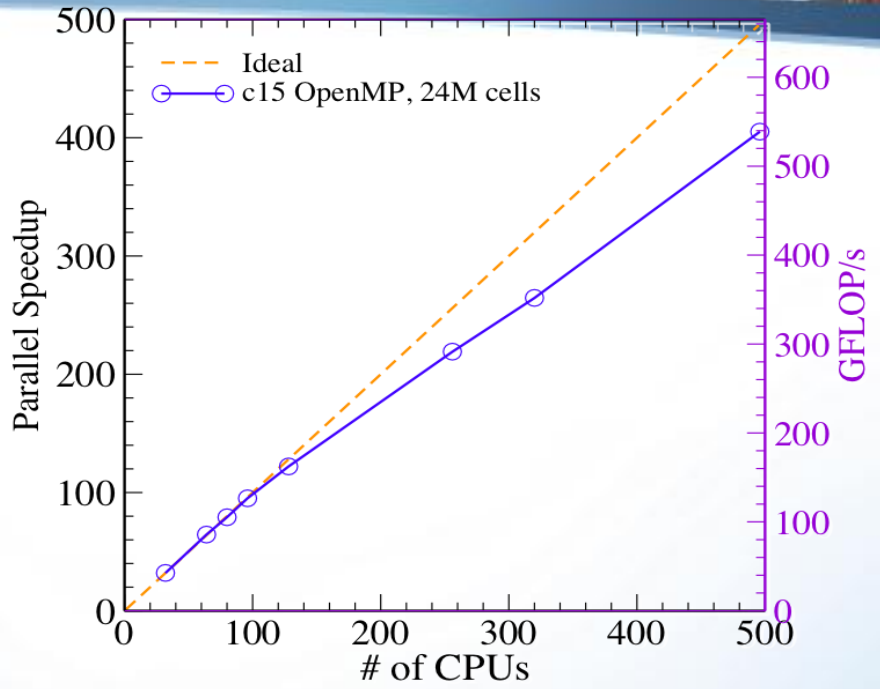

64
56
48
40
32
24
16
8
0
Speedup
Gustafson's Law(5%)
Gustafson's Law(50%)
Amdahl's Law(5%)
Amdahl's Law(50%)
0
8
16
24
32
40
48
56
64
N プロセッサコア


16
14
12
10
8
6
4
2
0
Speedup
0
2
4
6
8
10
12
14
16
Nプロセッサ

NASAによる流体解析コード



Virtual Flight on High-Performance Architectures
M. J. Aftosmis, S. M. Murman, M. Nemec, NASA Ames
SC2004,Pittsburgh, PA, Nov. 6-12, 2004
Graphics courtesy of NASA Ames

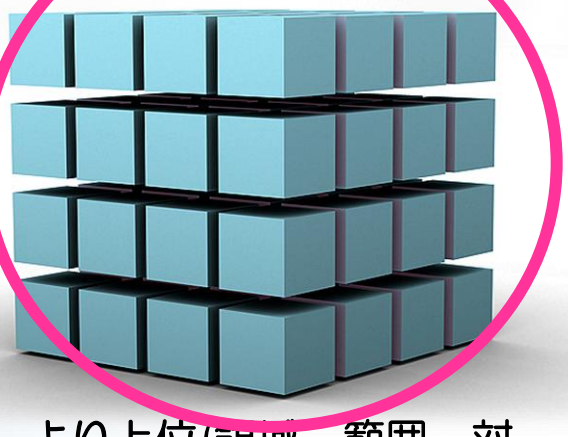


並列性能

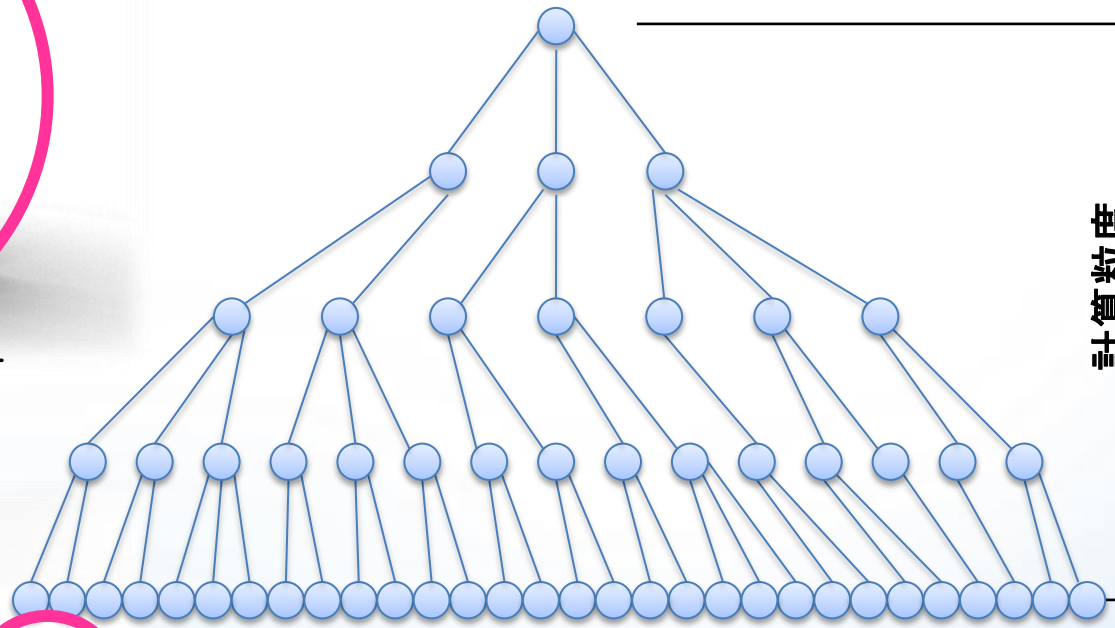
- 496プロセッサで405倍の性能向上が可能
  - 540 GFLOP/s
  - CPUあたりの性能：1.33 GFLOP/s
- 短時間でのシュミレーションを可能とし、問題への緊急的な対応を可能となります。

よりハイレベルでの並列化

全体処理の把握とその並列化の検討



より上位(領域、範囲、対象）での並列化



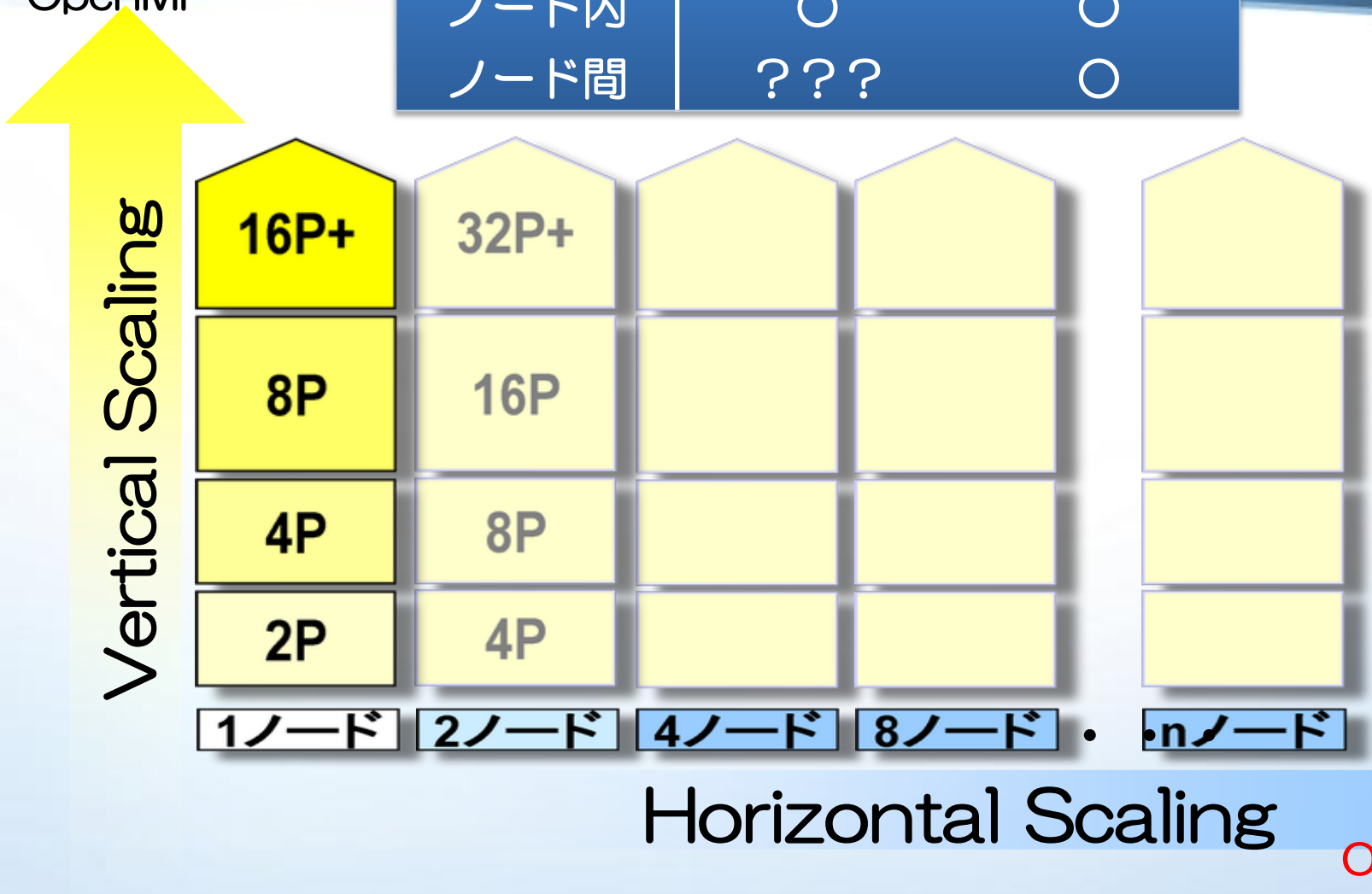
処理の末端での並列化

個々の処理の並列化の検討
コンパイラによる並列化（ベクトル化や自動並列化）は一般にはこのレベルでの並列化

大
困難・複雑
計算粒度
並列化
小
容易

シングルAPIでの並列処理

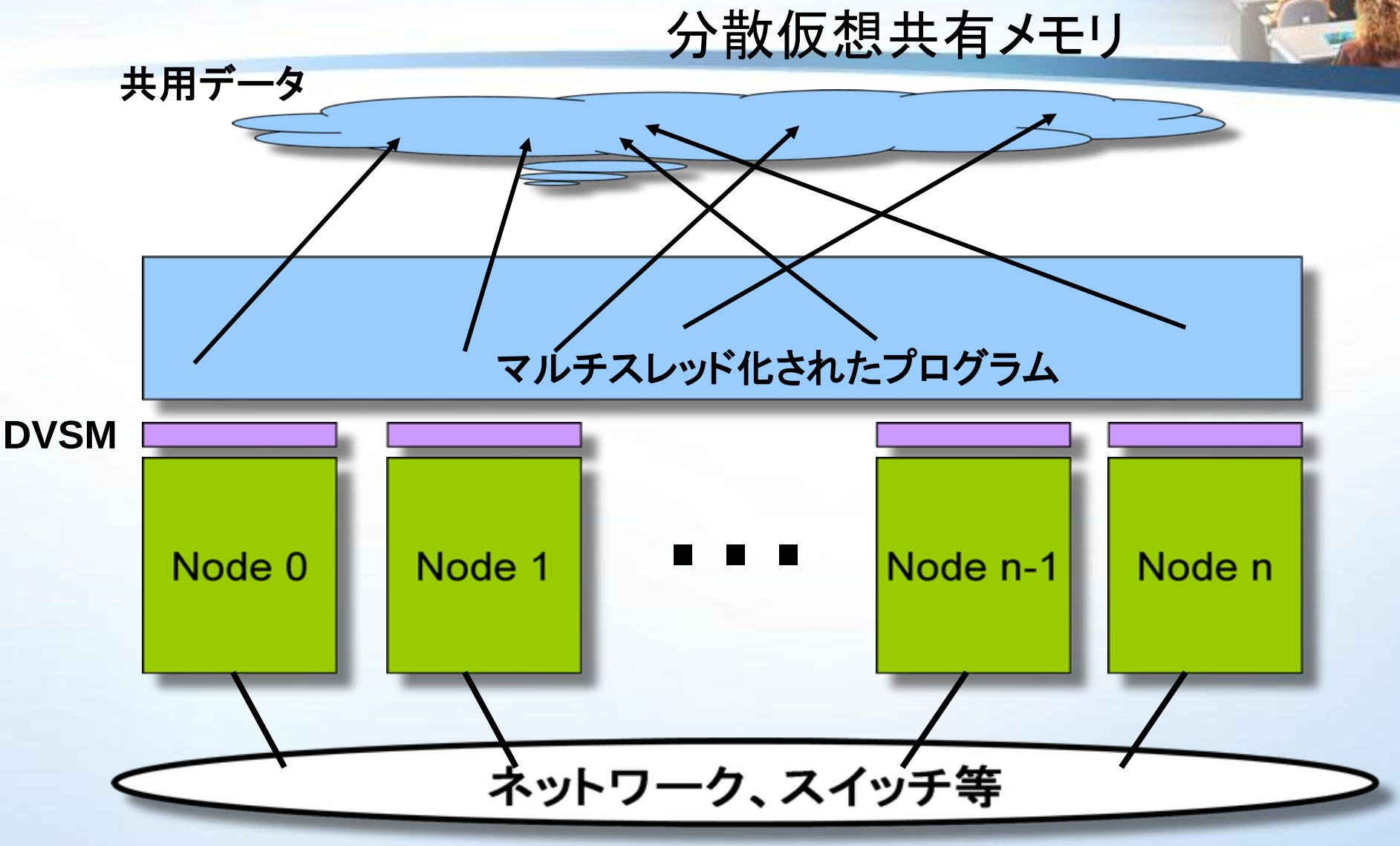
	OpenMP	MPI
ノード内	○	○
ノード間	???	○



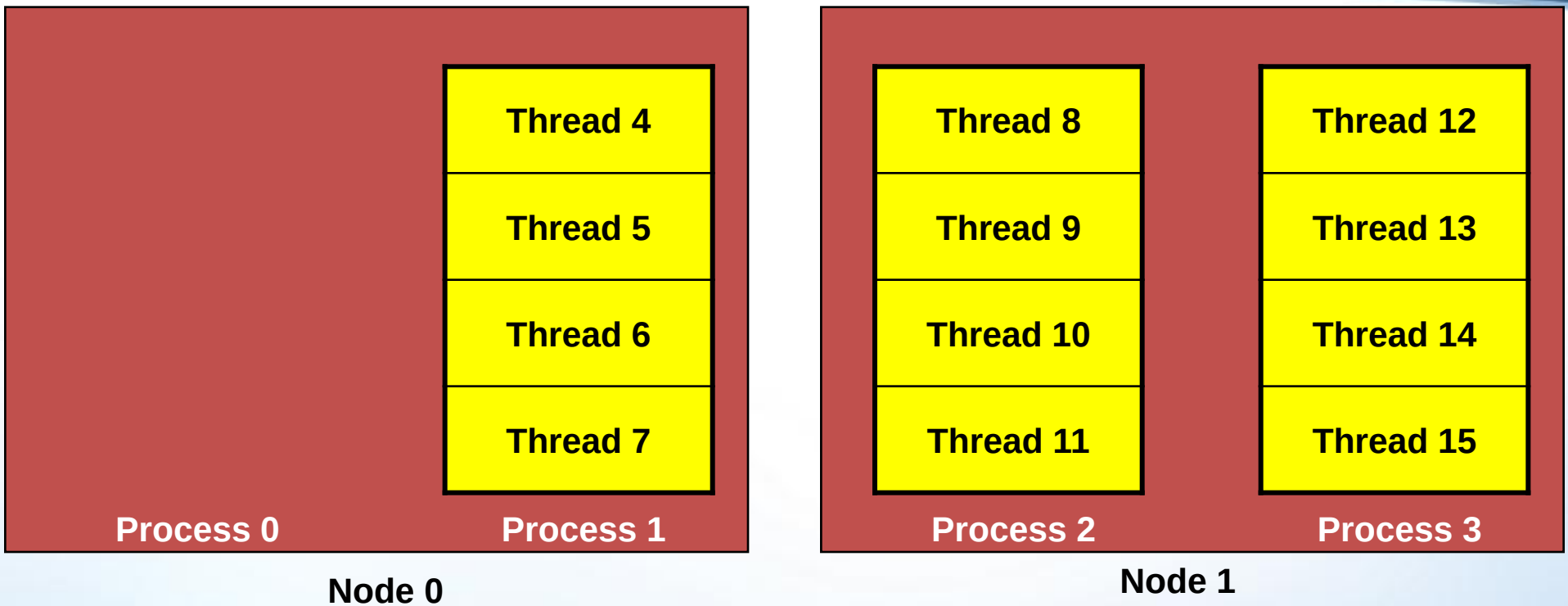
OpenMPの価値

- シュミレーションでのソフトウエア開発
  - 大規模なシュミレーションを行うアプリケーションは、C++、C、Fortranで記述されている
  - ソフトウエア開発と利用は数十年単位で継続して行われる
- クラスタやスーパーコンピュータ
  - MPIを利用した大規模並列処理が一般的
  - MPI利用の限界と課題
    - 全てのアプリケーションに適用出来る訳ではない
    - スケーラビリティは様々な制限を受ける
    - “many cores” への対応がMPIタイプのAPIでは非常に難しい
  - OpenMPはこのようなMPIに対する他の選択肢の提供と同時にMPIを補完する役割を担う（MPI + OpenMP)

インテルクラスタOpenMP



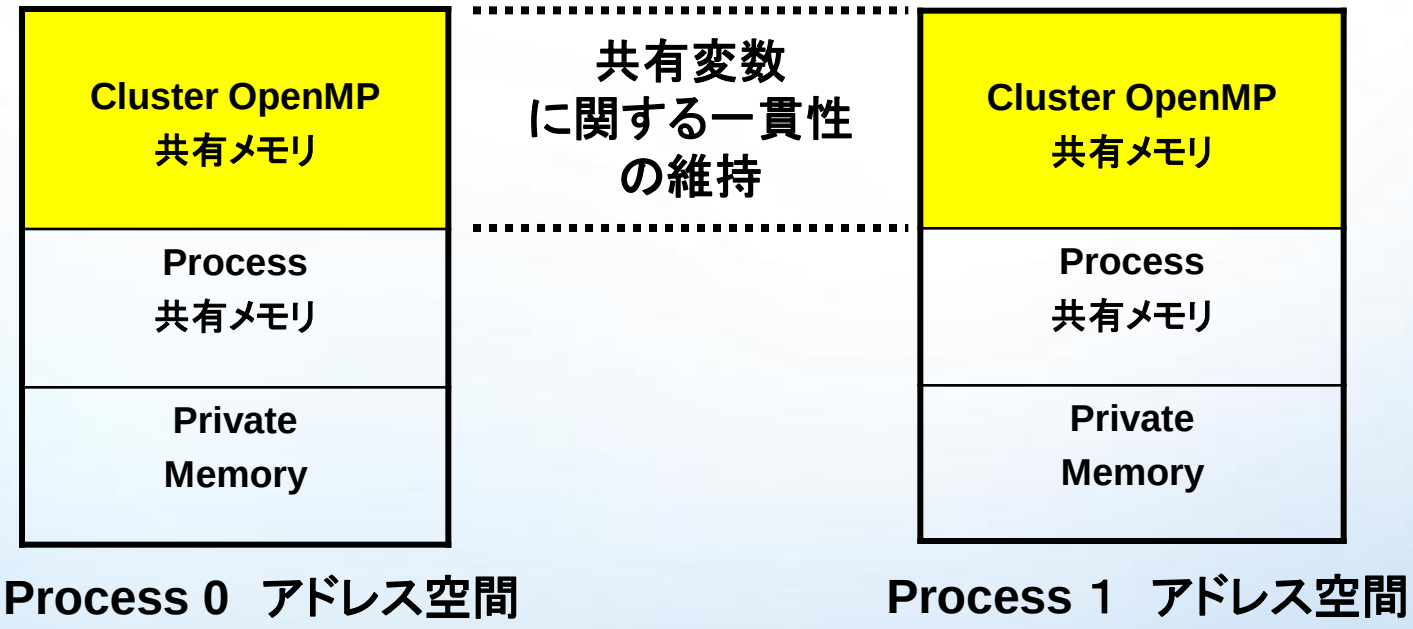
Cluster OpenMP スレッドとプロセス



Node – クラスタを構成する各計算機システム
プロセス – Linuxのプロセス
スレッド – OpenMPのスレッド（プロセス中のスレッド）

Cluster OpenMPメモリモデル

- プロセス間でOpenMPスレッドがアクセスする変数は、’sharable’ 変数として指示
  - 通常のOpenMPの共有データの宣言では、プロセス間でのデータ共有は出来ない（プロセス内でのデータの共有を宣言）



’sharable’ 変数の宣言

- OpenMPの指示句で、’shared’と指示される変数や共有される変数は、この ‘sharable’の宣言が必要（ただし、ファイルポインタなどのシステムが使用する変数は除く）
- ‘sharable’の宣言が必要な変数に関する情報は、コンパイラ時のメッセージとして、確認も可能

  例）–clomp-sharable-propagation オプションの指定

  ifort –cluster-openmp –clomp-sharable-propagation –ipo file.f file2.f

  fortcom: Warning: Sharable directive should be inserted by user as ‘!dir$ omp sharable(n)’ in file file.f, line 23, column 16
- データの‘sharable’ 宣言は、コンパイラ指示行で適用

```
#pragma intel omp sharable(var)      // C,C++
!dir$ omp sharable(var)              ! Fortran
```

’sharable’ 変数の宣言

オリジナルコード	コンパイラオプション	コンパイラオプションと同等の ’sharable’ 変数の宣言
common /blk/ a(100)	**-clomp-sharable-commons**	common /blk/ a(100) !dir$ omp sharable (/blk/)
real a(100) save a	**-clomp-sharable-localsaves**	real a(100) save a !dir$ omp sharable (a)
module m real a(100)	**-clomp-sharable-modvars**	module m real a(100) !dir$ omp sharable (a)

簡単なクラスタOpenMPプログラム例

```
#include <omp.h>

static int x;
#pragma intel omp sharable(x)

int main()
{
        x = 0;
    #pragma omp parallel shared(x)
    {
        #pragma omp critical
        x++;
    }
printf("%d should equal %d¥n", omp_get_max_threads(), x);
}
```

sharableディレクティブでコンパイラに変数xはDVSM上に置かなければならないことを指示する

クラスタOpenMPプログラム例

```
      common/storage/ x(2*nk), q(0:nq-1), qq(0:nq-1)
      data              dum /1.d0, 1.d0, 1.d0/
!dir$ omp sharable(/storage/)
!dir$ omp sharable(sx,sy)

!$omp parallel default(shared)
!$omp& private(k,kk,t1,t2,t3,t4,i,ik,x,x1,x2,l,qq)
      do 115 i = 0, nq - 1
         qq(i) = 0.d0
 115  continue

!$omp do reduction(+:sx,sy)
      do 150 k = 1, np
         kk = k_offset + k
.....
         do 140 i = 1, nk
   .....
                sx    = sx + t3
                sy    = sy + t4
             endif
 140     continue
 150  continue
!$omp end do nowait
```

sharableディレクティブでコンパイラに変数xはDVSM上に置かなければならないことを指示する

プログラムのコンパイルと実行

コンパイル時に、Cluster OpenMPでの並列処理を指定

```
$ icc –cluster-openmp test.c
$ cat kmp_cluster.ini
--hostlist=rufus,dufus --processes=2 --process_threads=4
$ a.out
8 should equal 8
```

設定ファイル kmp_cluster.ini に利用する2つのノードを指定し、各ノード上で利用するプロセス数と各プロセスあたりのスレッド数を指定する。この場合には、合わせて、8スレッドでの並列処理となる。

Cluster OpenMP プログラムのコンパイルと実行例

クラスタ間共有データの定義

OpenMP実行時関数

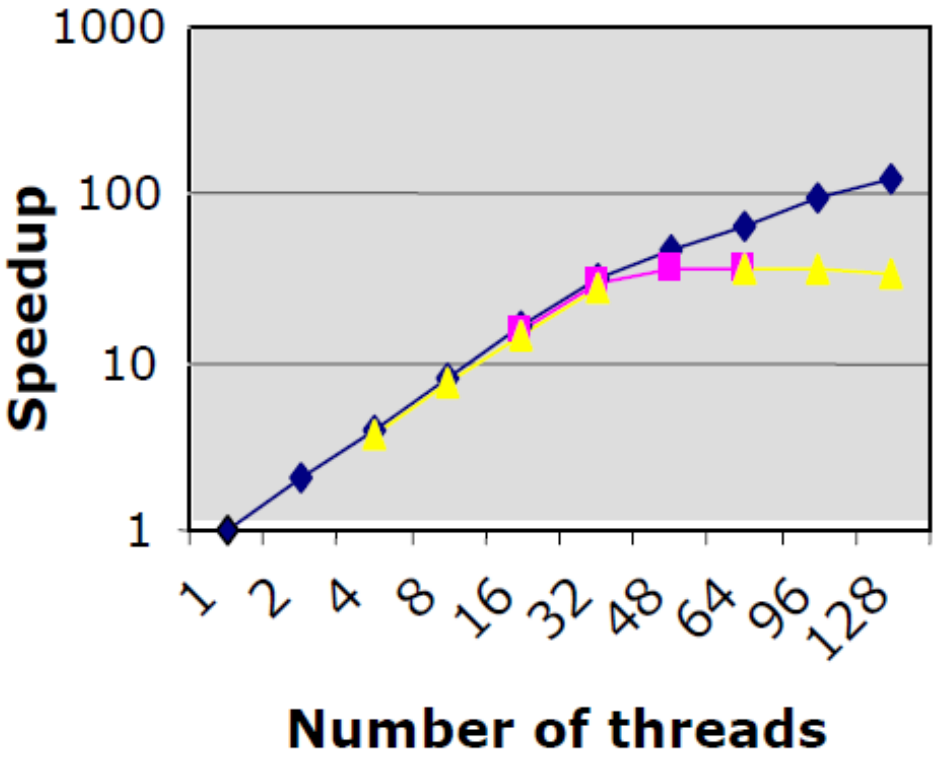
コンパイルとメッセージ

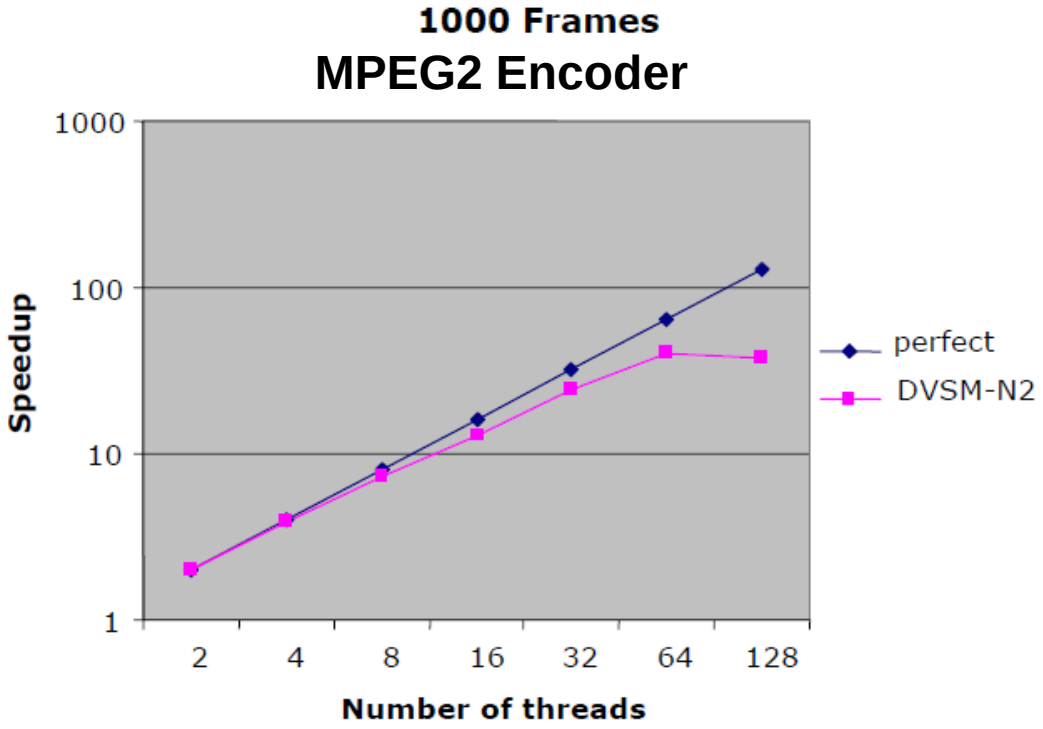
並列実行処理環境の設定

```
$ cat -n cpi.c
     1  #include <omp.h>                          // OpenMP実行時関数呼び出し
     2  #include <stdio.h>                        //  のためのヘッダファイルの指定
     3  #include <time.h>
     4  static int num_steps = 1000000;
     5  double step;
     6  #pragma intel omp sharable(num_steps)
     7  #pragma intel omp sharable(step)
     8  int main ()
     9  {
    10  int i, nthreads;
    11  double start_time, stop_time;
    12  double x, pi, sum = 0.0;
    13  #pragma intel omp sharable(sum)
    14  step = 1.0/(double) num_steps;            // OpenMPサンプルプログラム:
    15  #pragma omp parallel private(x)           // 並列実行領域の設定
    16    {
    17      nthreads = omp_get_num_threads();     // 実行時関数によるスレッド数の取得
    18  #pragma omp for reduction(+:sum)          // "for" ワークシェア構文
    19      for (i=0;i< num_steps; i++){          // privateとreduction指示句
    20        x = (i+0.5)*step; // の指定
    21        sum = sum + 4.0/(1.0+x*x);
    22        }
    23    }
    24      pi = step * sum;
    25      printf("%5d Threads : The value of PI is %10.7f¥n",nthreads,pi);
    26  }
    27
$ icc -cluster-openmp -O -xT cpi.c
cpi.c(18) : (col. 1) remark: OpenMP DEFINED LOOP WAS PARALLELIZED.
cpi.c(15) : (col. 1) remark: OpenMP DEFINED REGION WAS PARALLELIZED.
$ cat kmp_cluster.ini
--hostlist=node0,node1 --processes=2 --process_threads=2 --no_heartbeat --startup_timeout=500
$ ./a.out
4 Threads : The value of PI is  3.1415927
```

性能(ベンチマーク)データ

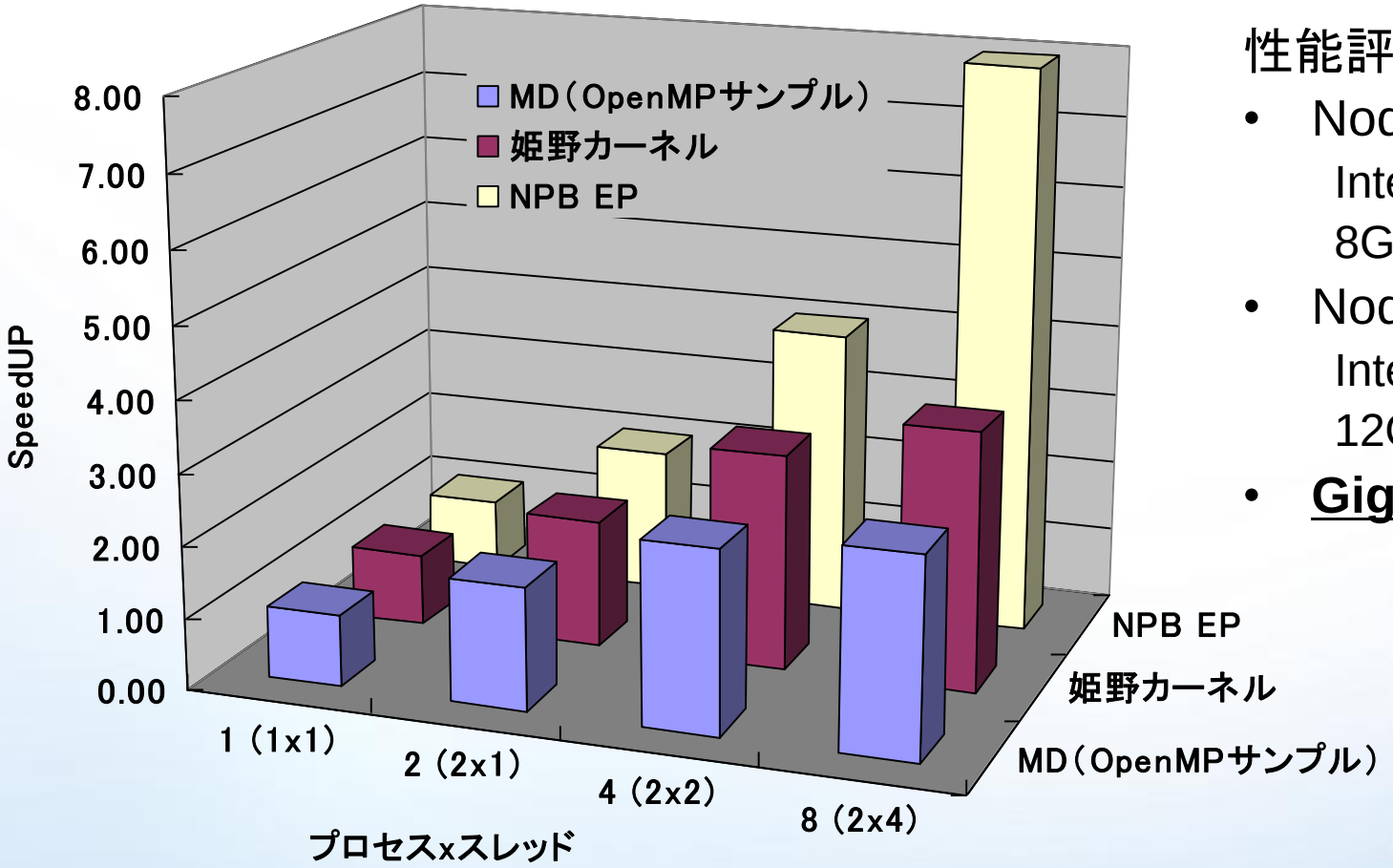
- 幾つかのベンチマークを実施した結果がインテルから報告されています。
- Data MiningやRenderingではある程度のスケーラビリティが示されています





インテル社のCluster OpenMP関連の資料より抜粋
Parallel Parallel Programming for Programming for Hybrid Hybrid Architectures Architec
Tom Lehmann
Technical Director Technical Director
HPC Programs Office HPC Programs Office
January 23, 2006

簡単なカーネルでの性能

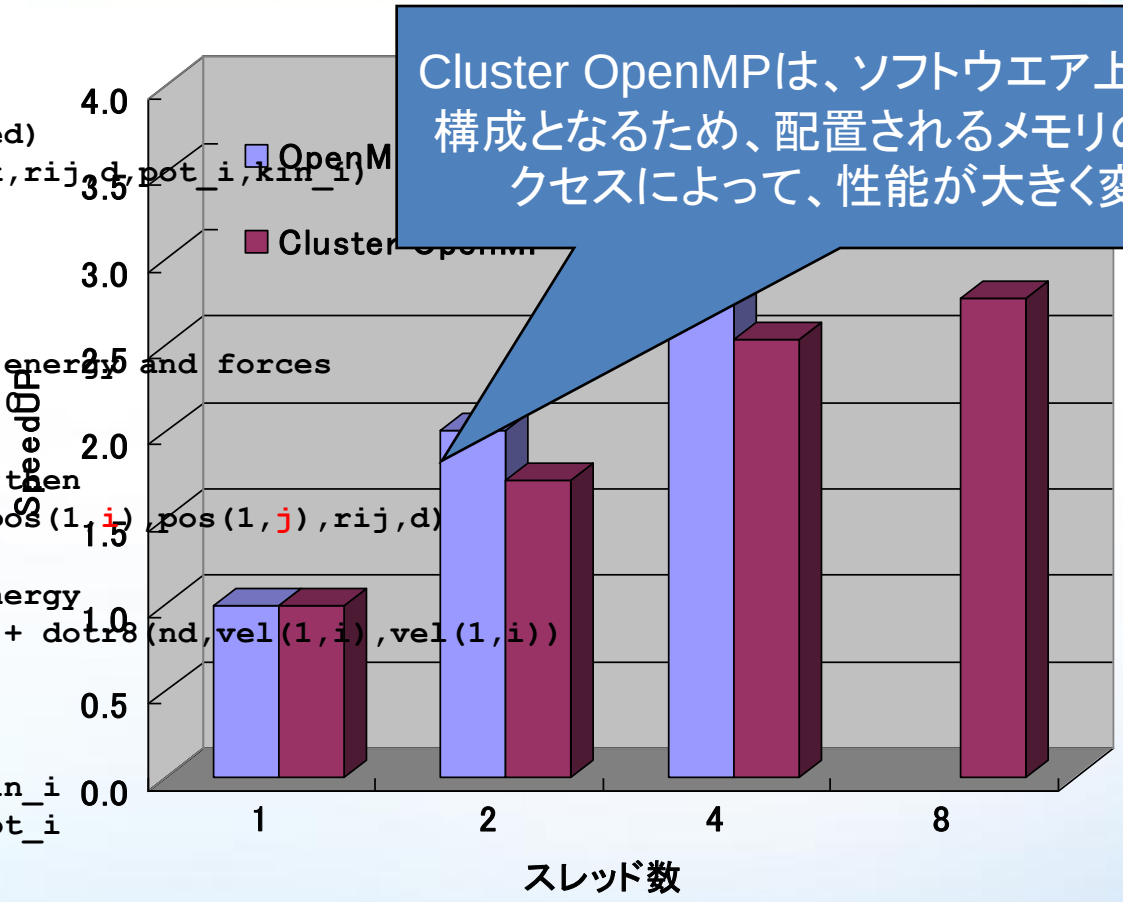


性能評価システム

- Node1
  Intel Xeon 5100 2.67GHz
  8GB memory
- Node2
  Intel Xeon 5100 2.67GHz
  12GB memory
- **Gigabit Ethernet**での接続

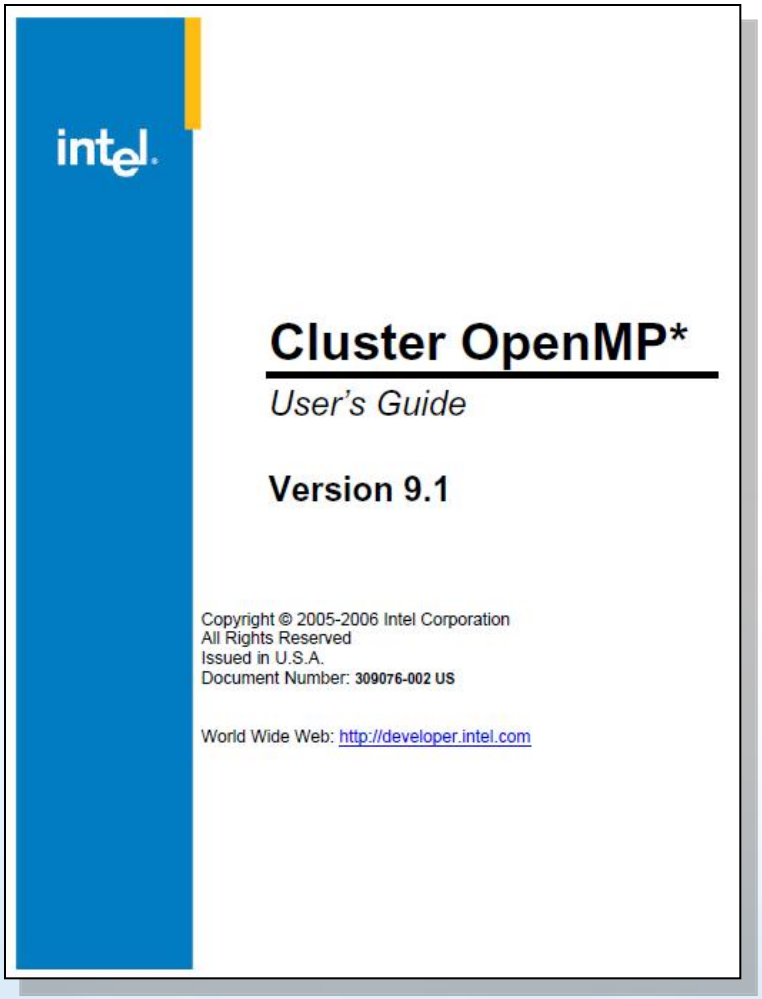
Cluster OpenMPでの注意点

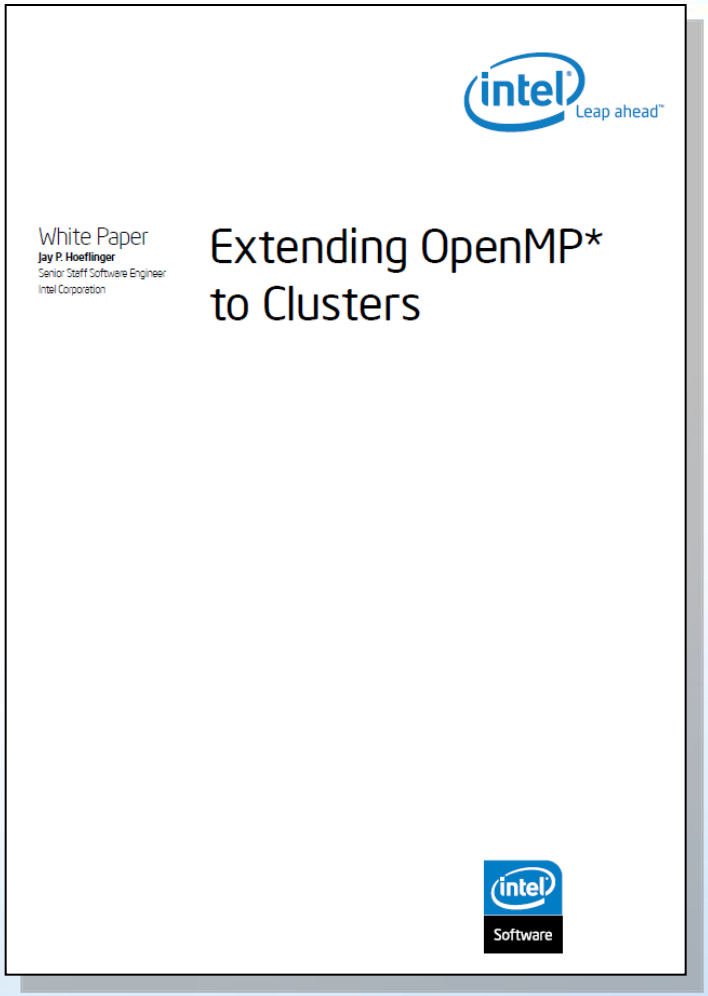
```
!$omp  parallel
!$omp& default(shared)
!$omp& private(i,j,k,rij,d,pot_i,kin_i)
      pot_i = 0.0
      kin_i = 0.0
!$omp  do
      do i=1,np
! compute potential energy and forces
      f(1:nd,i) = 0.0
      do j=1,np
      if (i .ne. j) then
      call dist(nd,pos(1,i),pos(1,j),rij,d)
.......
! compute kinetic energy
      kin_i = kin_i + dotr8(nd,vel(1,i),vel(1,i))
      enddo
!$omp  end do
!$omp  critical
      kin = kin + kin_i
      pot = pot + pot_i
!$omp  end critical
!$omp  end parallel
```



Cluster OpenMPは、ソフトウエア上は、‘NUMA’の構成となるため、配置されるメモリの場所とそのアクセスによって、性能が大きく変わります。

更に詳しい情報


intel
Cluster OpenMP*
User's Guide
Version 9.1
Copyright © 2005-2006 Intel Corporation
All Rights Reserved
Issued in U.S.A.
Document Number: 309076-002 US
World Wide Web: http://developer.intel.com


intel
Leap ahead
White Paper
Jay P. Hoeflinger
Senior Staff Software Engineer
Intel Corporation
Extending OpenMP* to Clusters
intel
Software

一般的OpenMPの課題

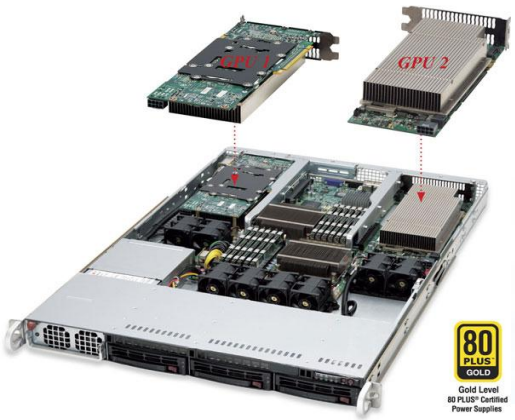
- OpenMP版のコンパイル時の問題
  - OpenMP構文に基づく並列化によって、マイクロプロセッサ向け最適化が阻害される
- 実行時ライブラリでのオーバヘッド
  - 頻繁なライブラリ呼び出しの悪影響
- アルゴリズムの変更のオーバヘッド
  - プログラム並列化のためのコードの冗長化やコードの追加
- 同期処理
  - Fork-Join モデルによる過大な同期処理
  - 負荷分散
- メモリ階層の有効活用
  - キャッシュ、ローカルメモリ、リモートメモリの参照頻度

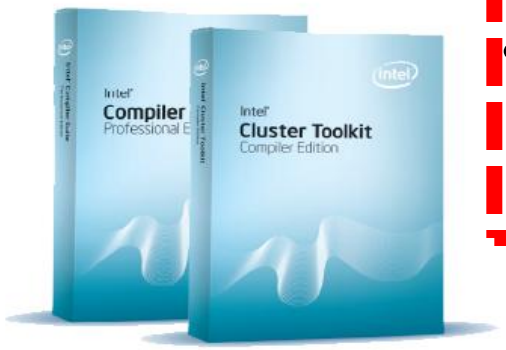
OpenMPの課題

- ハードウエアの動向
  - 今後は複数ソケットの製品はすべてNUMAアーキテクチャ
- OpenMP 3.0リリース
  - NUMA対応の拡張無し
  - アフィニティ問題
    - データの配置及び利用とスレッド実行の管理

OpenMP 3.0: The World is still flat, no support for cc-NUMA (yet)!
http://terboven.wordpress.com/category/openmp/

ハイブリッド:現代のトレンド？

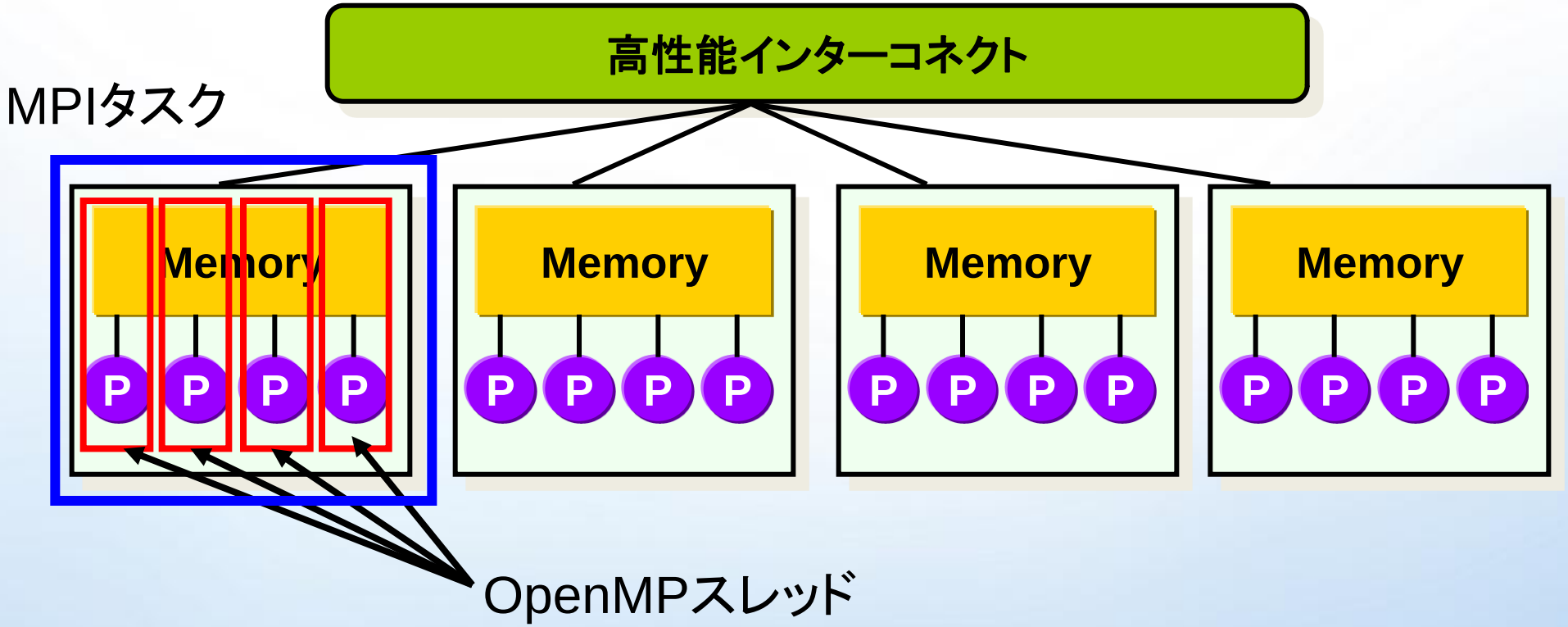




- ハイブリッドカー
  - 内燃機関動力（ガソリンエンジンやディーゼルエンジン）と蓄電池
- ハイブリッドコンピューティング
  - GPU＋CPUによるハイパフォーマンスコンピューティング
- ハイブリッド並列プログラミング
  - スレッドプログラング＋MPI

MPI/OpenMPハイブリッドモデル

- MPIでは領域分割などの疎粒度での並列処理を行う
- OpenMPは、各MPIタスク内で、ループの並列化などのより細粒度での並列化を担う
- 計算は、タスク-スレッドの階層構造を持つ



MPI/OpenMPハイブリッドコード

- MPIで並列化されたアプリケーションにOpenMPでの並列化を追加
- MPI通信とOpenMPでのワークシェアを利用して効率良い並列処理の実現

Fortran

```
include ‘mpif.h’
program hybsimp


call MPI_Init(ierr)
call MPI_Comm_rank (...,irank,ierr)
call MPI_Comm_size (...,isize,ierr)
! Setup shared mem, comp. & Comm
!$OMP parallel do
      do i=1,n
        <work>
      enddo
! compute & communicate
call MPI_Finalize(ierr)
end
```

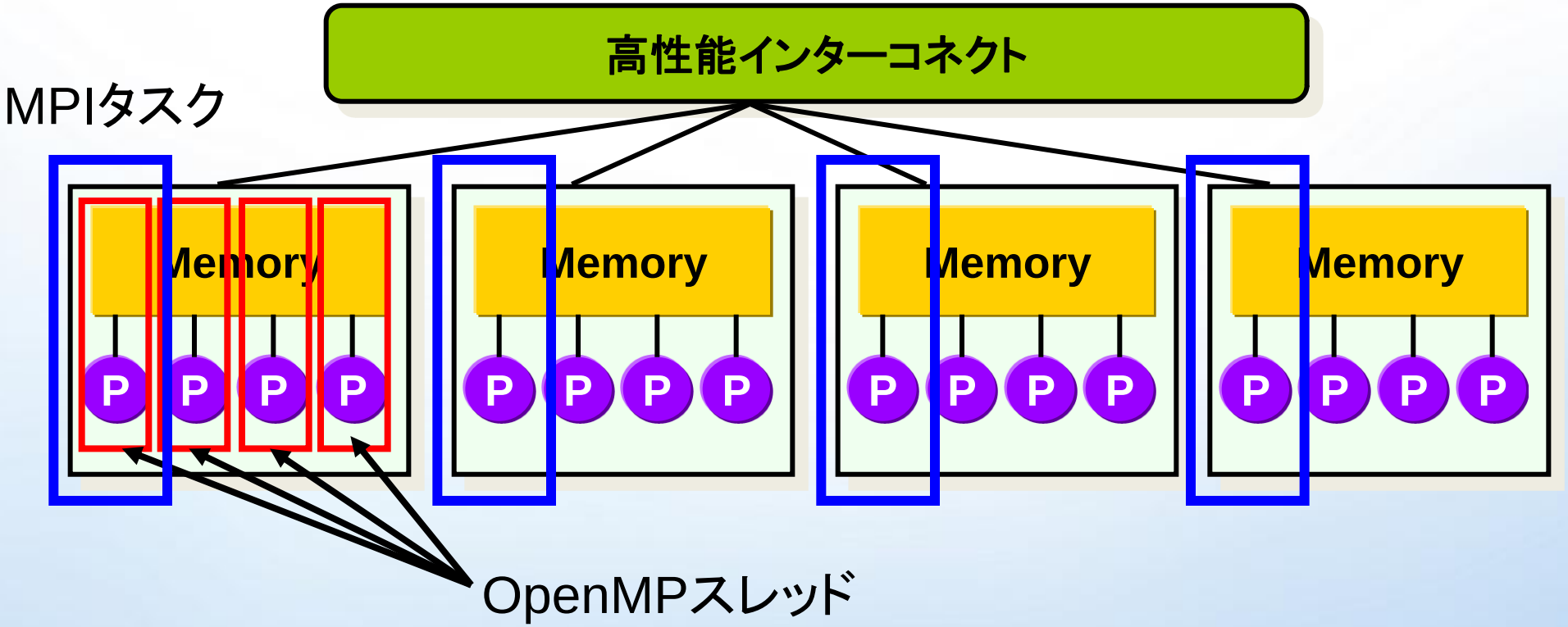
C/C++

```
#include <mpi.h>
int main(int argc, char **argv){
int rank, size, ierr, i;


ierr= MPI_Init(&argc,&argv[]);
ierr= MPI_Comm_rank (...,&rank);
ierr= MPI_Comm_size (...,&size);
//Setup shared mem, compute & Comm
#pragma omp parallel for
   for(i=0; i<n; i++){
      <work>
   }
// compute & communicate
ierr= MPI_Finalize();
```

OpenMP/MPIハイブリッドモデル

- MPIは実績のある高性能な通信ライブラリ
- 計算と通信を非同期に実行することも可能
- 通信はマスタースレッド、シングルスレッド、全スレッドで実行することが可能



OpenMP/MPIハイブリッドコード

- OpenMPのプログラムにMPI通信を追加
- 既存のOpenMPプログラムの拡張やスレッドプログラムの新規開発時のオプションとして選択
- MPIは非常に高速また最適化されたデータ通信ライブラリ

Fortran

```
include ‘mpif.h’
program hybmas

!$OMP parallel

  !$OMP barrier
  !$OMP master
  call MPI_<Whatever>(…,ierr)
  !$OMP end master
  !$OMP barrier



!$OMP end parallel
end
```

C/C++

```
#include <mpi.h>
int main(int argc, char **argv){
int rank, size, ierr, i;

#pragma omp parallel
{
  #pragma omp barrier
  #pragma omp master
  {
  ierr=MPI_<Whatever>(…)
  }
  #pragma omp barrier


}
```

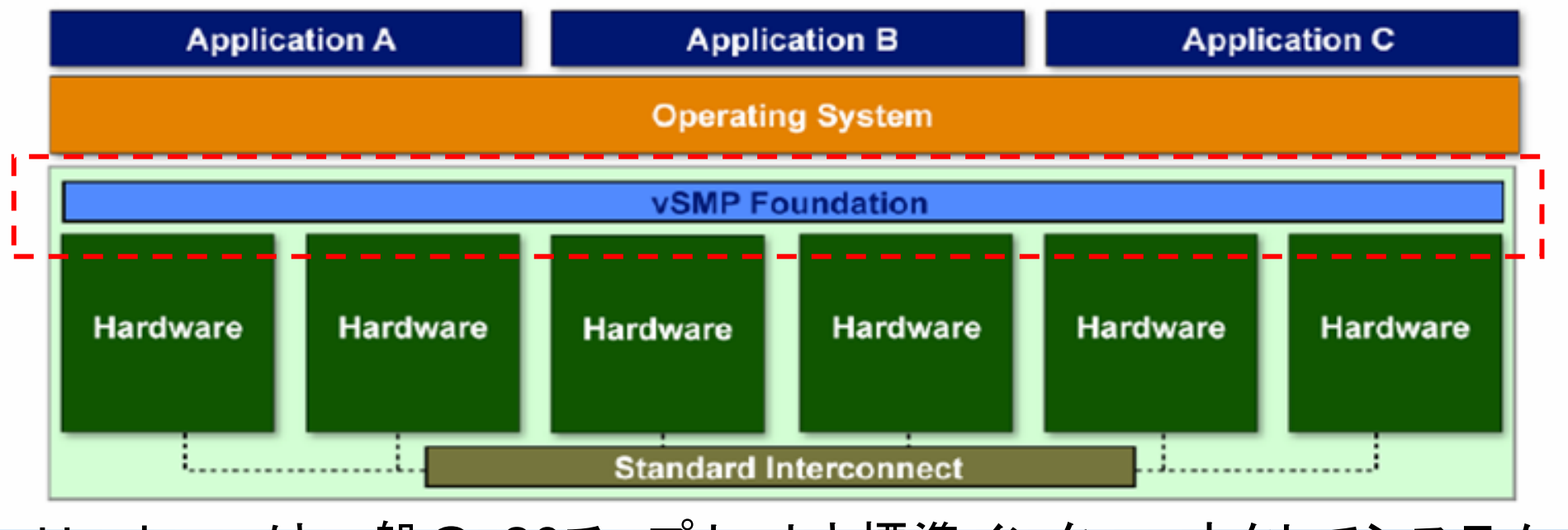
ハイエンド仮想化


複数の仮想マシン
仮想化ソフトウエア
サーバ（仮想化なし）
アプリケーション
オペレーティングシステム
サーバ仮想化
ハイエンド仮想化
一台の仮想マシン
アプリケーション
アプリケーション
アプリケーション
アプリケーション
アプリケーション
オペレーティングシステム
仮想化ソフトウエア

ScaleMP vSMPアーキテクチャ

アプリケーションについては、他のx86システムと100%のバイナリ互換を実現

OSは通常のLinuxディストリビューションが利用可能



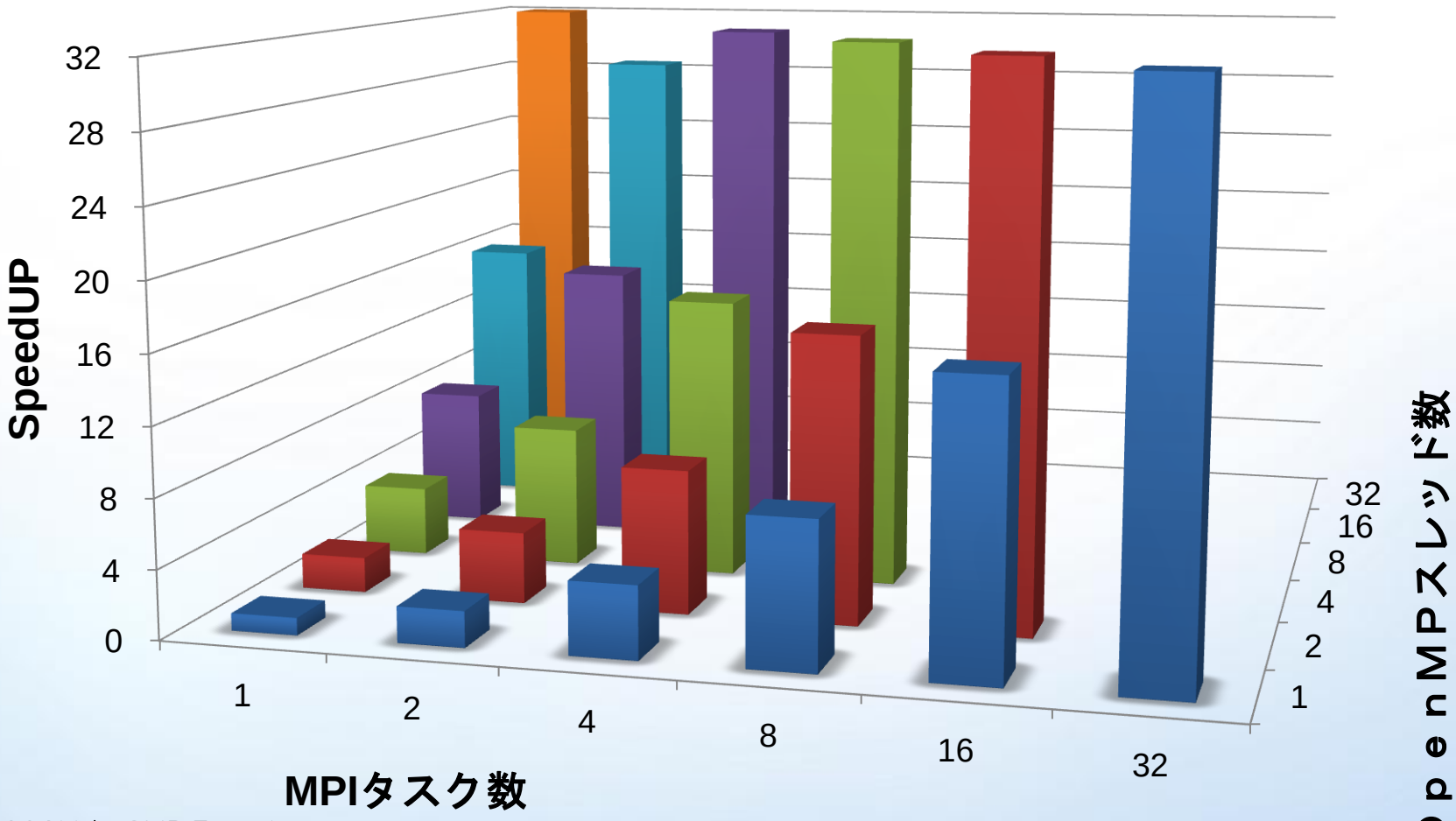
Hardwareは一般のx86チップセットと標準インターコネクトでシステムの構築が可能

vSMP Foundation でのシステムのSMP拡張を実現

OpenMP/MPI/ハイブリッド

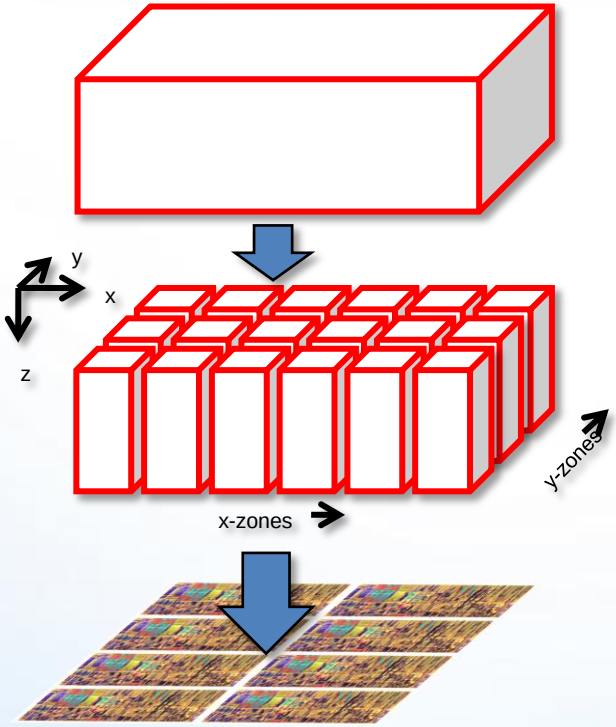
Hybrid OpenMP MPI Benchmarkproject (″homb″)

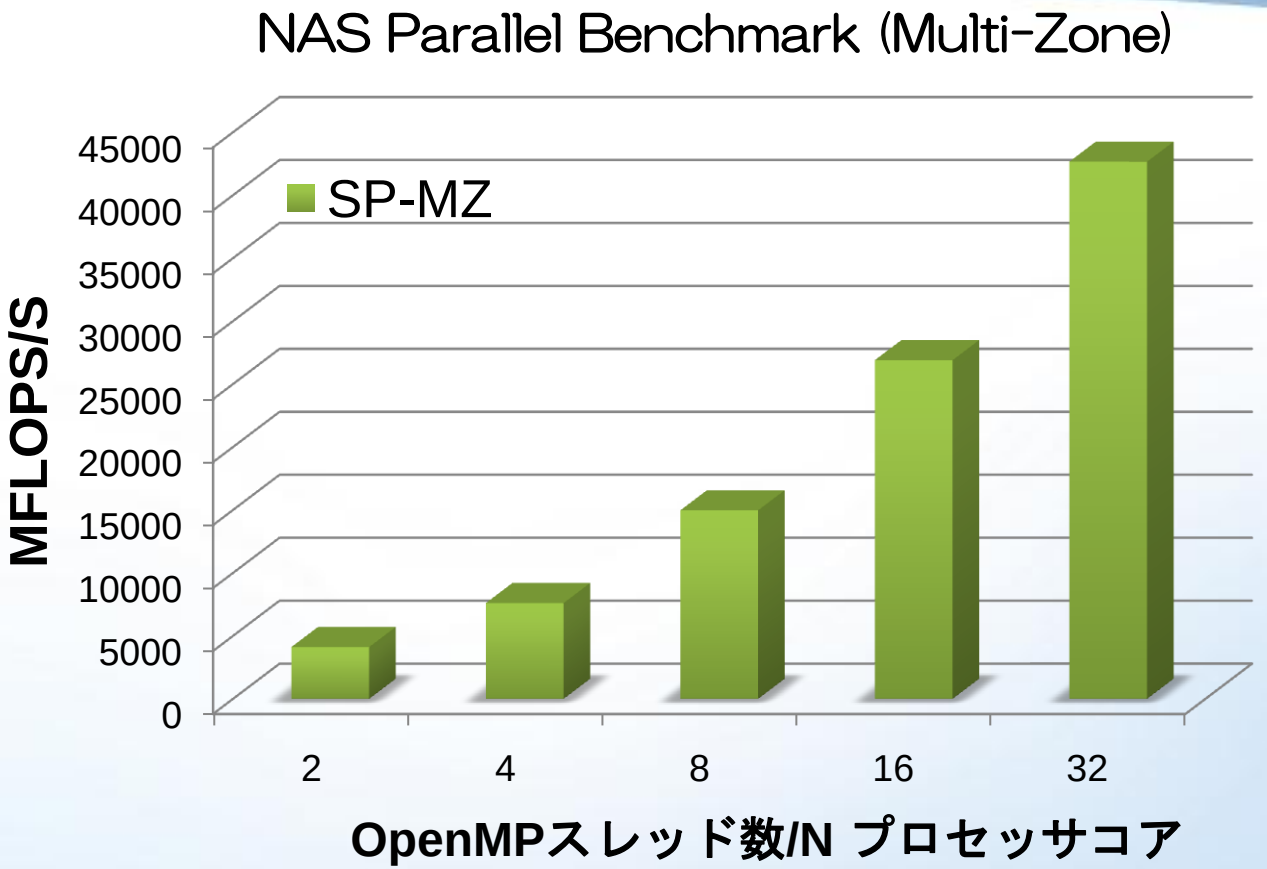
This is the Hybrid OpenMP MPI Benchmarkproject (″homb″)
This project was registered on SourceForge.net on May 16, 2009, and is described by the project team as follows:
HOMB is a simple benchmark based on a parallel iterative Laplace solver aimed at comparing the performance of MPI, OpenMP, and hybrid codes on SMP and multi-core based machines.



Xeon 5550 (2.66GHz) vSMP Foundation

OpenMPベンチマーク

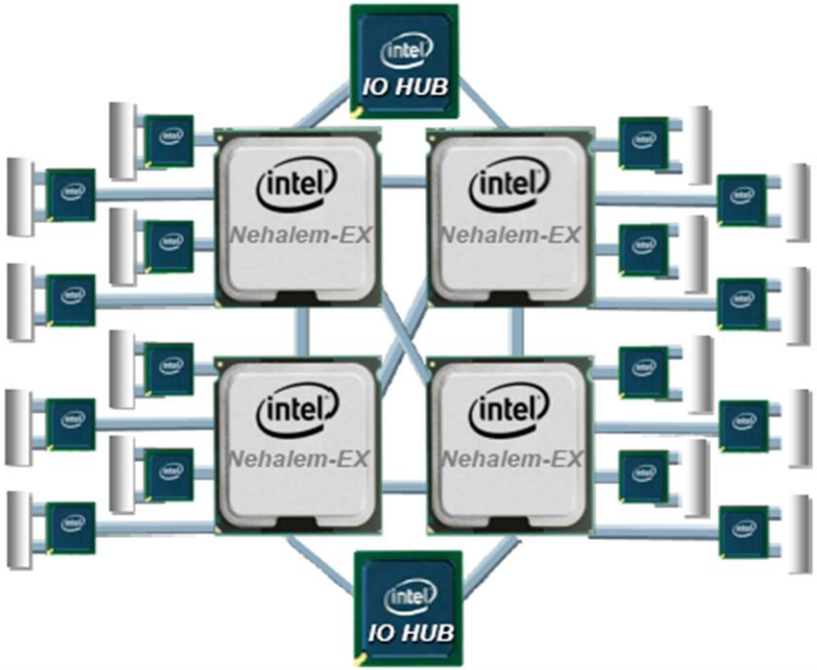




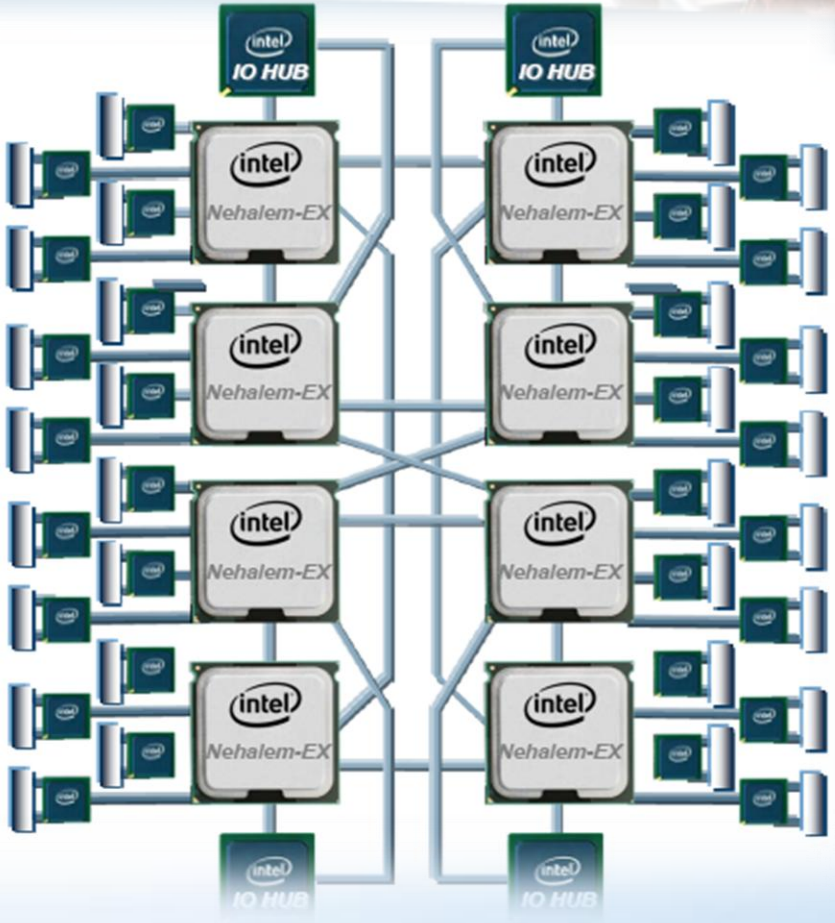
著名な公開ベンチマークツールである NAS Parallel Benchmark (NPB) の一つであるNPB-MZ (NPB Multi-Zone)はより粒度の大きな並列化の提供を行っています。NPB-MZでは、ハイブリッド型の並列処理やネストしたOpenMPのテストが可能です。ここでの結果は、OpenMPだけでの並列処理の性能を評価しています。
Xeon 5550 (2.66GHz) vSMP Foundation

Nehalem-EX トポロジ

IDF2009
INTEL DEVELOPER FORUM

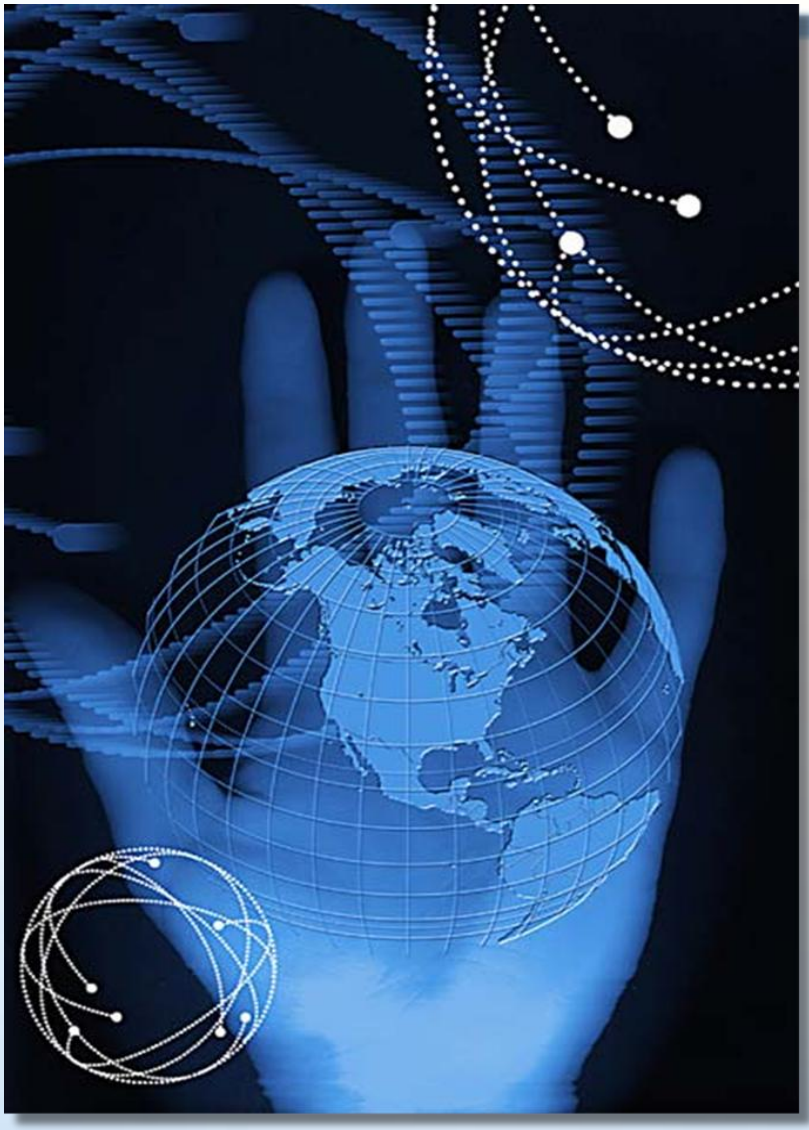


4プロセッサトポロジ
32プロセッサコア



8プロセッサトポロジ
64プロセッサコア

この資料について



お問い合わせ
0120-090715
携帯電話・PHSからは（有料）
03-5875-4718
9:00-18:00 (土日・祝日を除く)

WEBでのお問い合わせ
www.sstc.co.jp/contact

1/17/2010

スケーラブルシステムズ株式会社