High Performance and Productivity

並列プログラミング課題と挑戦

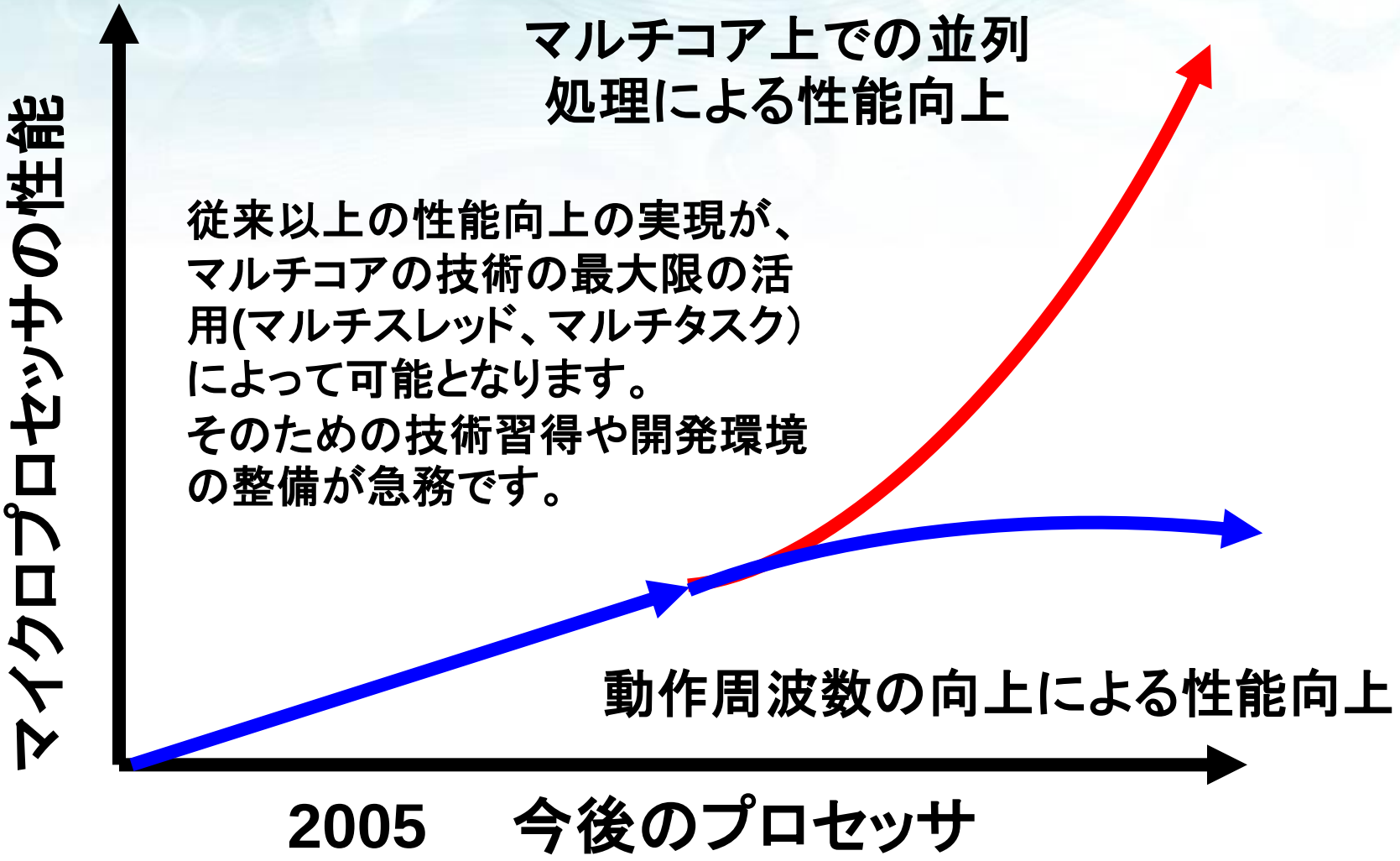
スケーラブルシステムズ株式会社

HPCシステムの利用の拡大の背景

- シュミレーションへの要求
  - より複雑な問題をより精度良くシュミレーションすることが求められている
- HPCシステムでの並列処理の要求の拡大
  1. モデル、アルゴリズム、解析対象は何れもより複雑で、規模の大きなものになっている
  2. マイクロプロセッサのマルチコア化
  3. クラスタに代表される並列計算機システムの低コスト化

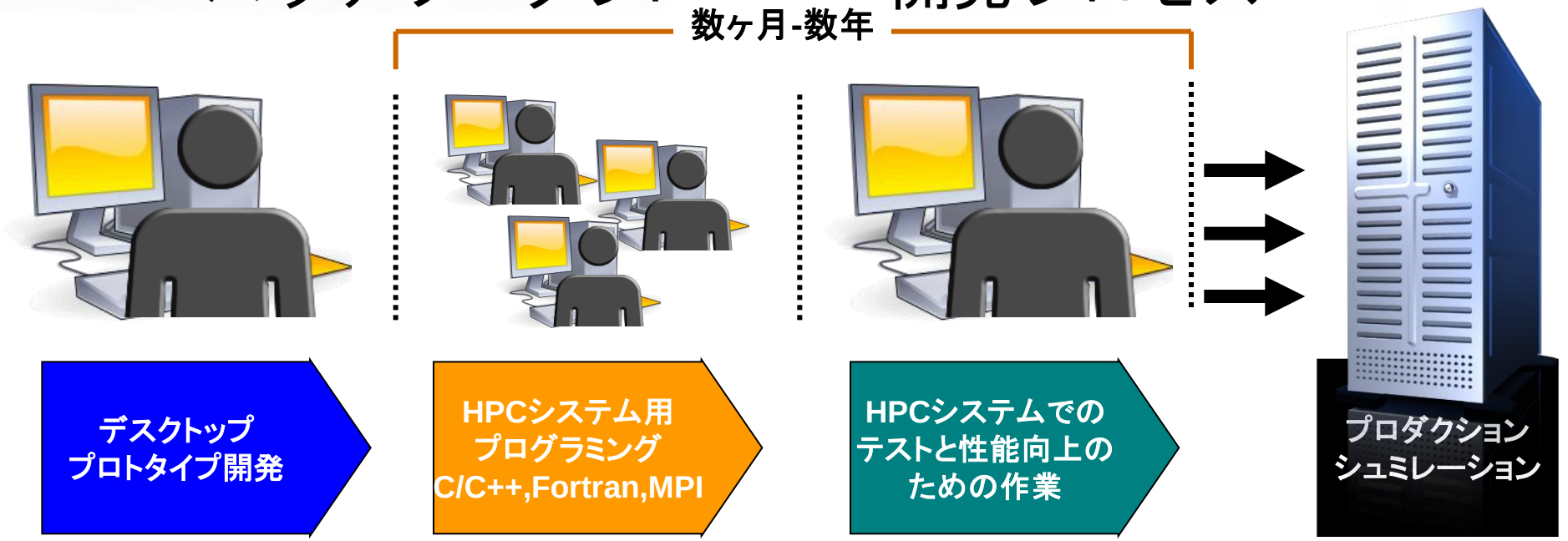
HPCシステムの一般化

マイクロプロセッサの性能向上
動作周波数からマルチコアへ



並列アプリケーション

- 従来型の並列アプリケーションの開発プロセス
  - バッチワークフローでの開発プロセス

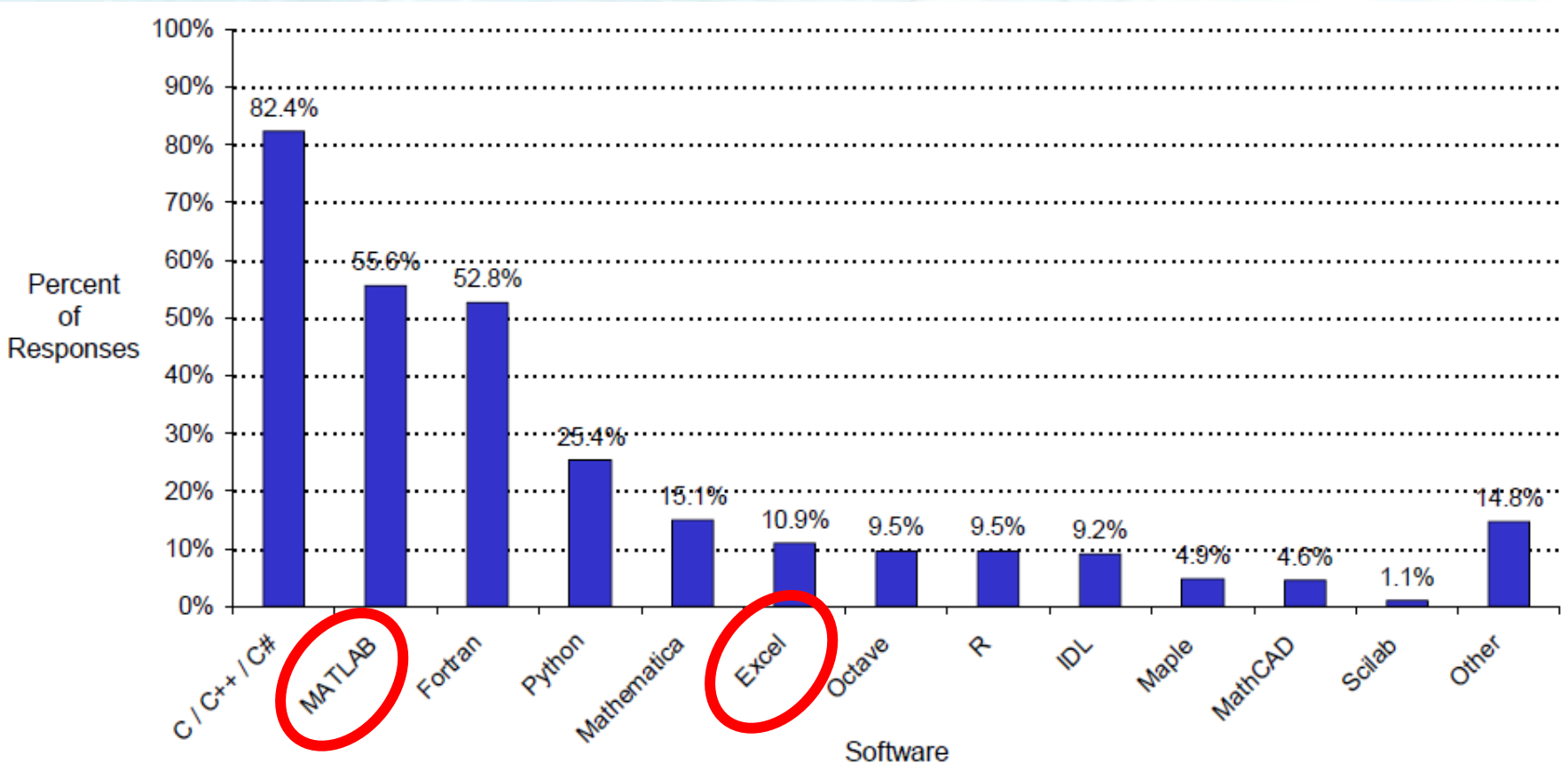


デスクトップ

- デスクトップでのプロトタイプ計算
  - ハイレベル言語（例えば、MATLABなど）
  - マイクロソフト エクセルなどでの処理
- ユーザ
  - システムアーキテクチャやマイクロプロセッサのマイクロアーキテクチャを意識することなくプログラミングを行う
  - GUIを利用した開発環境

並列プログラミングでの課題

PC上でのプログラミングAPI



The Development of Custom Parallel Computing Applications
Simon Management Group
September 2006

複数選択可

並列アプリケーション

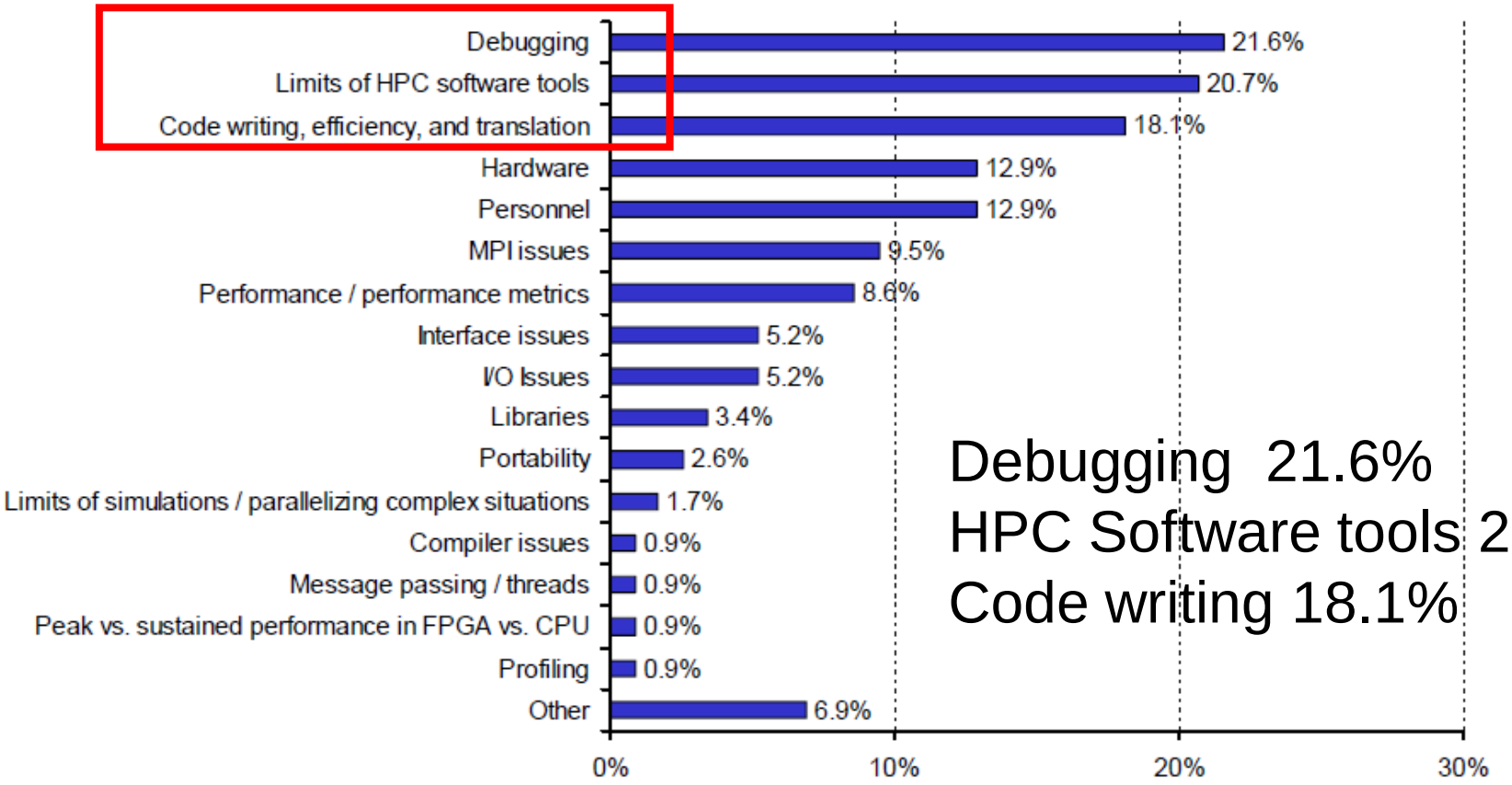
- プログラミング
  - 専門的な知識と並列APIに関する学習
  - C,Fortranなどのコンパイラを利用し、並列処理には、MPI（Message Passing Interface)などのコミュニケーションを明示的に記述
- ユーザ
  - 利用するHPCシステムのアーキテクチャを意識したプログラミング
  - プログラムの開発時にHPCシステムを利用する場合、バッチなどにジョブを投入し、プログラミングの確認を行うことが必要
  - 実際のモデル化やアルゴリズムを実際の解析対象で確認するのはプログラムの完成時まで困難

プログラミングの生産性

- 開発サイクル
  - プログラムのコーディング以外にも様々な作業が必要
- プログラムのデバッグ
  - 実際の解析モデルと入力データが必要
  - 逐次処理や対話処理が必要
- スケーラビリティの実現
  - HPCシステムでの高いスケーラビリティの実現には、高度なプログラミングが必要
  - アルゴリズムの選択やデータのモデル化の検討

並列プログラミングでの課題

- 並列アプリケーション開発でのボトルネック
  - デバッグ環境や開発ツールに関する不満



Debugging 21.6%
HPC Software tools 20.7%
Code writing 18.1%

The Development of Custom Parallel Computing Applications
Simon Management Group
September 2006

複数選択可

並列プログラミング

並列コンパイラ

並列デバッガ

並列数学ライブラリ

並列コード最適化ツール

スレッド解析ツール

最適化ツール

MPIタスク解析ツール

最適化ツール

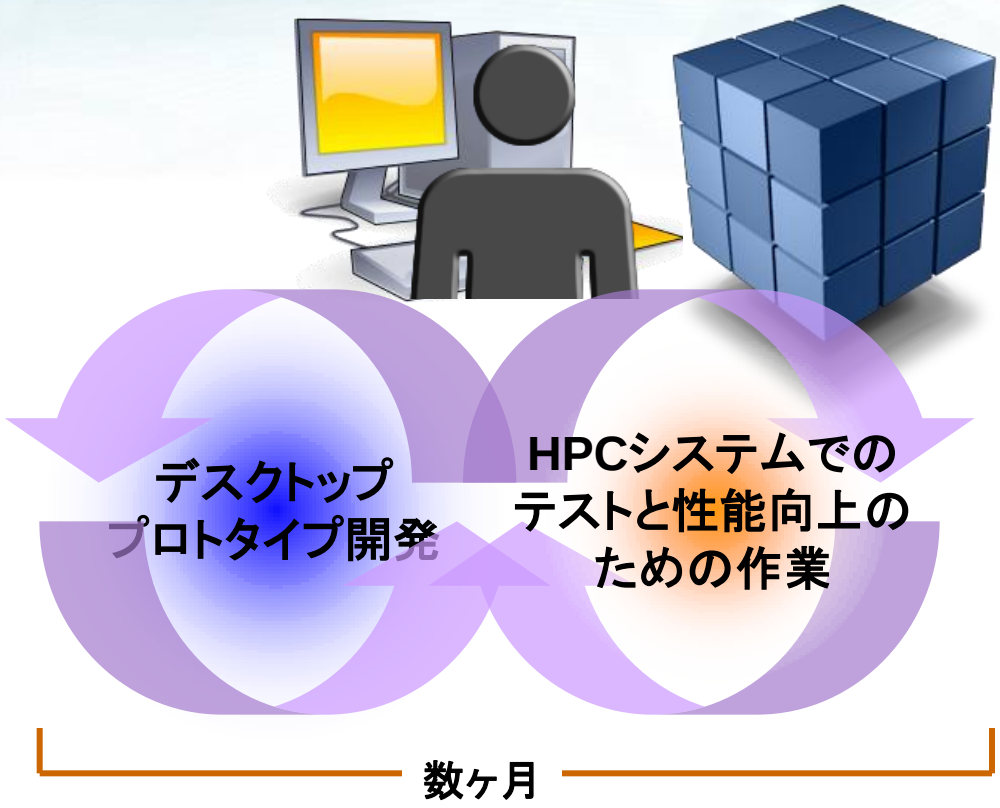
統合インターフェイス

ソフトウエアのギャップの解決

プログラミングのギャップ

プログラミングの生産性の向上

スケーラブルSMPシステム



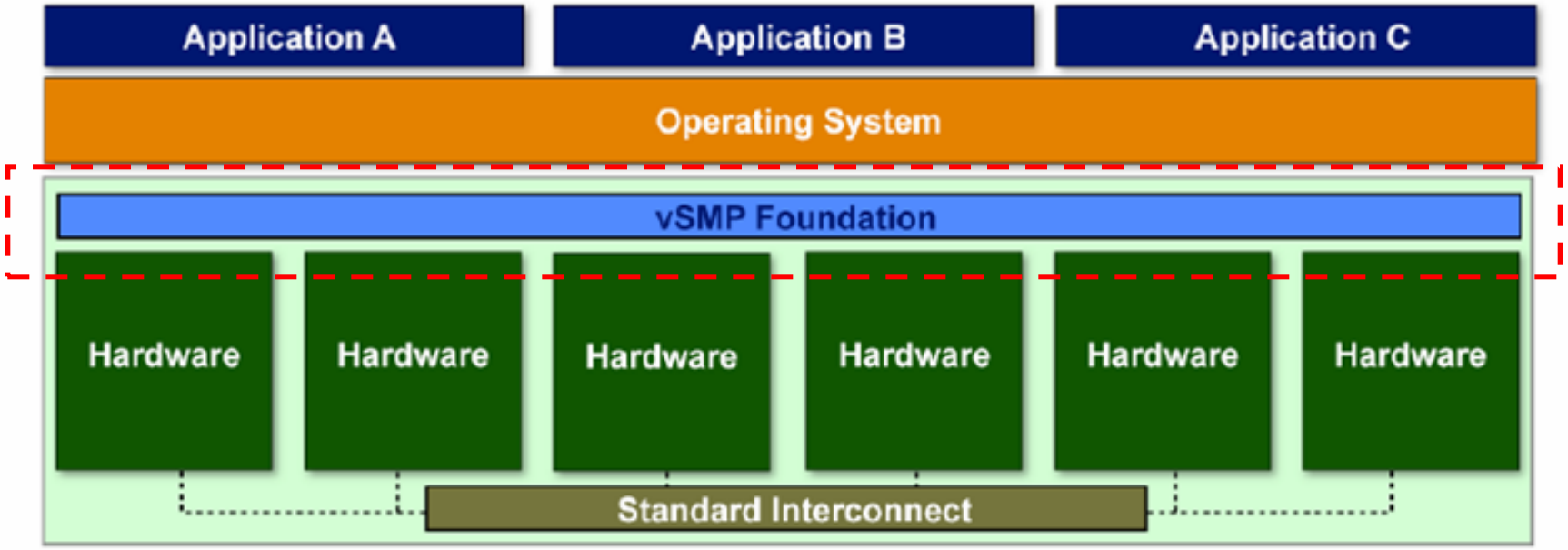
ハイエンド仮想化


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
複数の仮想マシン
仮想化ソフトウエア
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

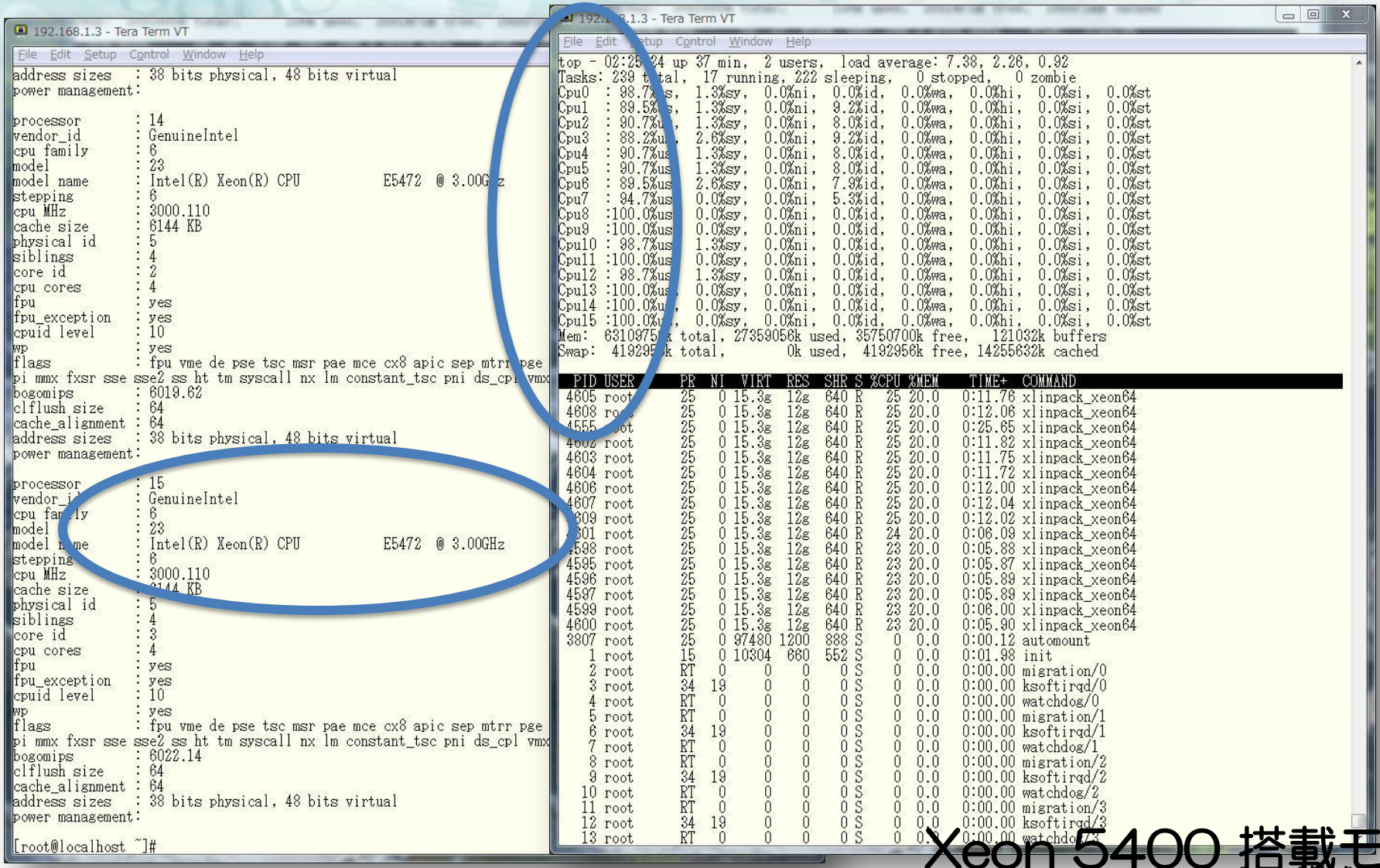
VXSMP 1440 システム

VXSMP 1440システムとは？

- VXPRO R1440ベースのvSMPシステム
- 1Uの筐体に4台のXeon 55xx/54xx/52xx を搭載し、最大16コアSMPを実現
- 96GBの共有メモリ空間（最大）
- 4台のHDDをOSが共有（最大6TB)

写真はXeon 5400 搭載モデル

VXSMP1400



Xeon 5400 搭載モデ

スケーラブルシステムズ株式会社

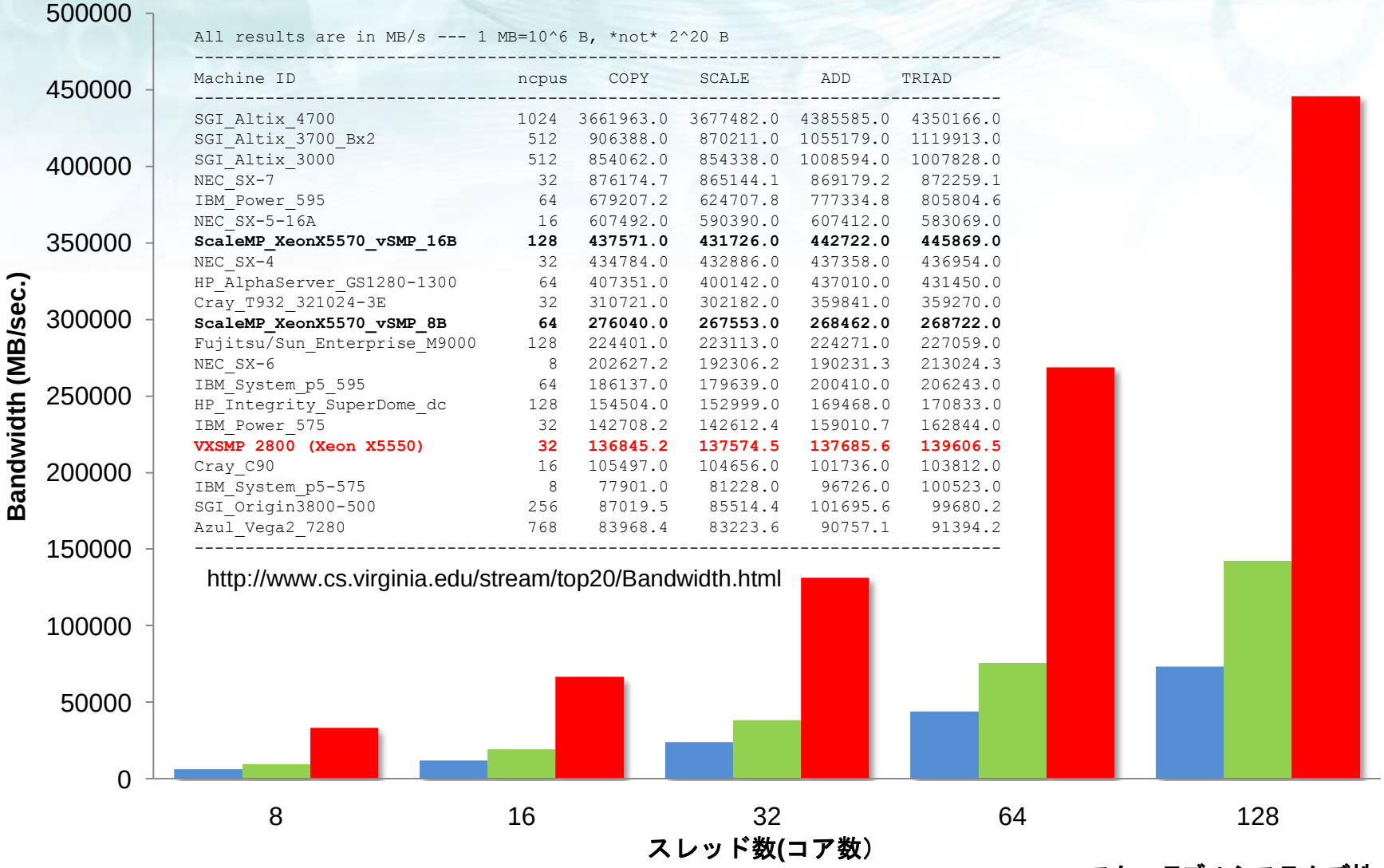
Stream (OMP)ベンチマーク

■ 1333MHz FSB(128cores/16 boards) ■ 1600MHz FSB(128cores/16 boards) ■ 6.4GT/s QPI(128cores/16 boards)

All results are in MB/s --- 1 MB=10^6 B, *not* 2^20 B

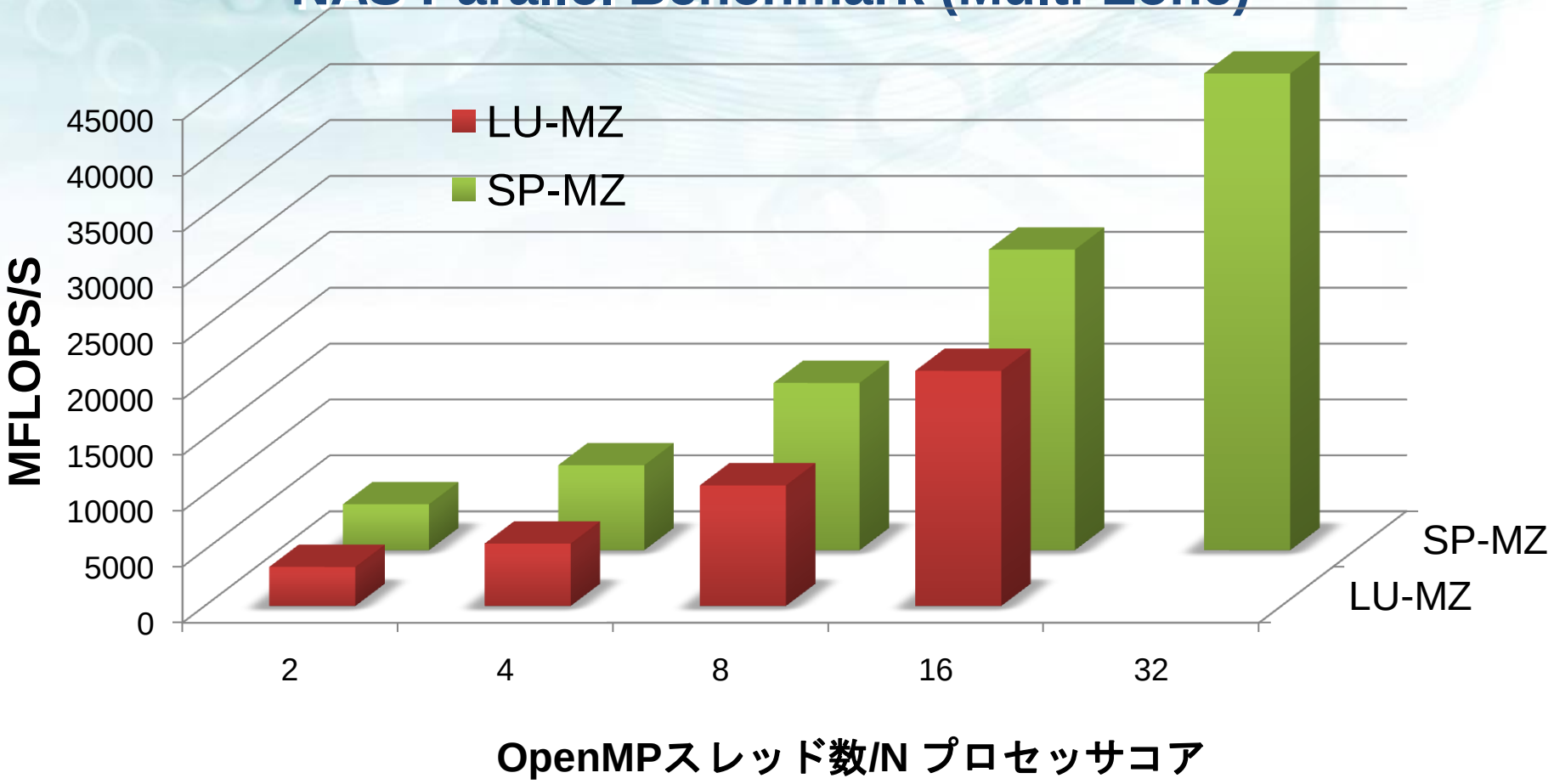
Machine ID	ncpus	COPY	SCALE	ADD	TRIAD
SGI_Altix_4700	1024	3661963.0	3677482.0	4385585.0	4350166.0
SGI_Altix_3700_Bx2	512	906388.0	870211.0	1055179.0	1119913.0
SGI_Altix_3000	512	854062.0	854338.0	1008594.0	1007828.0
NEC_SX-7	32	876174.7	865144.1	869179.2	872259.1
IBM_Power_595	64	679207.2	624707.8	777334.8	805804.6
NEC_SX-5-16A	16	607492.0	590390.0	607412.0	583069.0
ScaleMP_XeonX5570_vSMP_16B	**128**	**437571.0**	**431726.0**	**442722.0**	**445869.0**
NEC_SX-4	32	434784.0	432886.0	437358.0	436954.0
HP_AlphaServer_GS1280-1300	64	407351.0	400142.0	437010.0	431450.0
Cray_T932_321024-3E	32	310721.0	302182.0	359841.0	359270.0
ScaleMP_XeonX5570_vSMP_8B	**64**	**276040.0**	**267553.0**	**268462.0**	**268722.0**
Fujitsu/Sun_Enterprise_M9000	128	224401.0	223113.0	224271.0	227059.0
NEC_SX-6	8	202627.2	192306.2	190231.3	213024.3
IBM_System_p5_595	64	186137.0	179639.0	200410.0	206243.0
HP_Integrity_SuperDome_dc	128	154504.0	152999.0	169468.0	170833.0
IBM_Power_575	32	142708.2	142612.4	159010.7	162844.0
VXSMP 2800 (Xeon X5550)	**32**	**136845.2**	**137574.5**	**137685.6**	**139606.5**
Cray_C90	16	105497.0	104656.0	101736.0	103812.0
IBM_System_p5-575	8	77901.0	81228.0	96726.0	100523.0
SGI_Origin3800-500	256	87019.5	85514.4	101695.6	99680.2
Azul_Vega2_7280	768	83968.4	83223.6	90757.1	91394.2

http://www.cs.virginia.edu/stream/top20/Bandwidth.html



OpenMPベンチマーク

NAS Parallel Benchmark (Multi-Zone)



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。

OpenMP プログラム コンパイルと実行例

```
$ cat -n pi.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 = 1000000000;
     5  double step;
     6  int main ()
     7  {
     8          int i, nthreads;
     9          double start_time, stop_time;
    10          double x, pi, sum = 0.0;
    11          step = 1.0/(double) num_steps;              // OpenMPサンプルプログラム：
    12          #pragma omp parallel private(x)             // 並列実行領域の設定
    13          {       nthreads = omp_get_num_threads();   // 実行時関数によるスレッド数の
取得
    14                  #pragma omp for reduction(+:sum)    // "for" ワークシェア構文
    15                  for (i=0;i< num_steps; i++){        // privateとreduction指示
句
    16                          x = (i+0.5)*step;           // の指定
    17                          sum = sum + 4.0/(1.0+x*x);
    18                  }
    19          }
    20          pi = step * sum;
    21          printf("%5d Threads : The value of PI is %10.7f¥n",nthreads,pi);
    22  }
$ icc -O -openmp pi.c
pi.c(14) : (col. 3) remark: OpenMP DEFINED LOOP WAS PARALLELI
pi.c(12) : (col. 2) remark: OpenMP DEFINED REGION WAS PARALLELIZED.
$ setenv OMP_NUM_THREADS 2
$ a.out
    2 Threads : The value of PI is  3.1415927
```

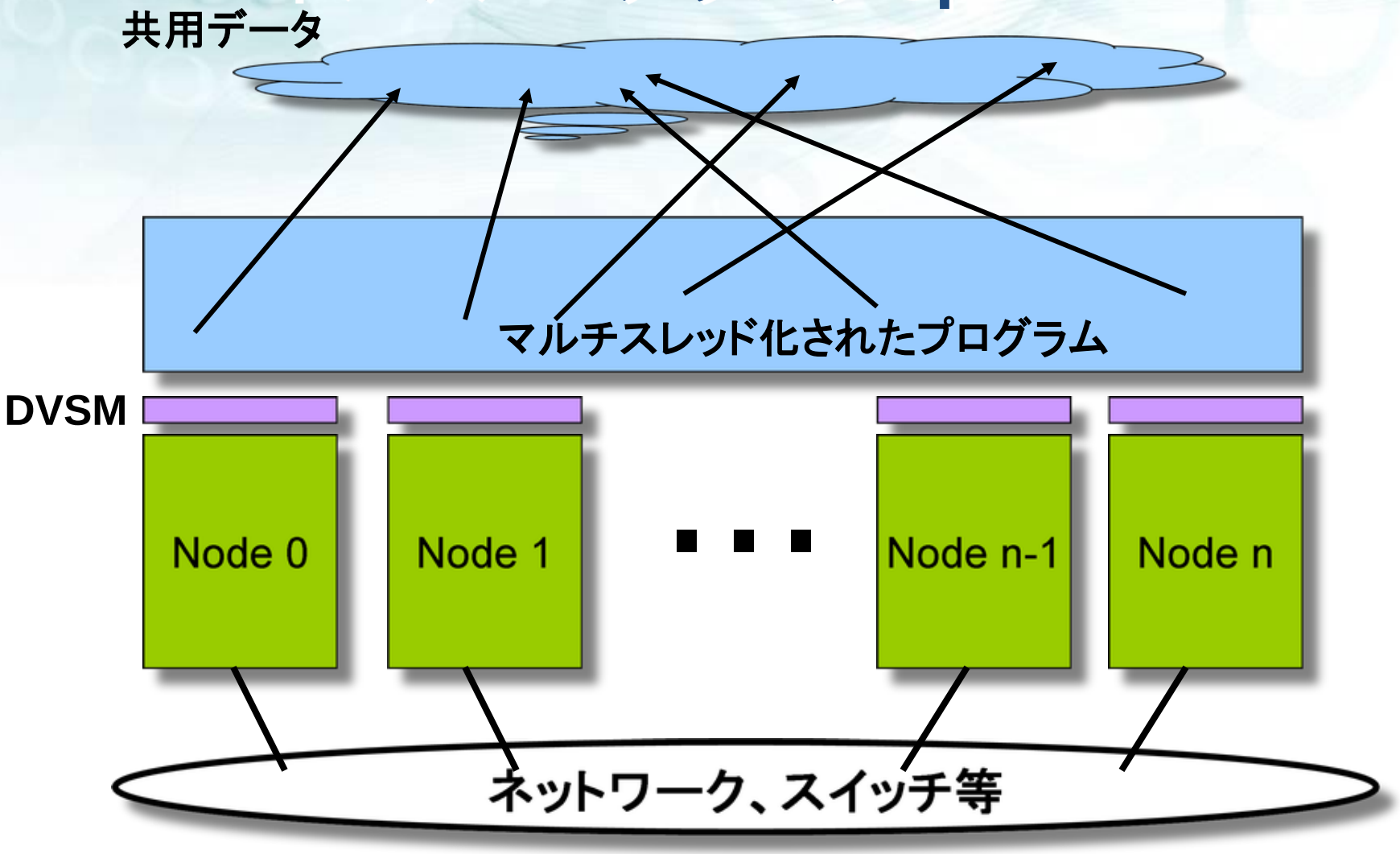
OpenMP指示行

OpenMP実行時関数

コンパイルとメッセージ

環境変数の設定

分散仮想共有メモリ（DVSM)
インテル　クラスタOpenMP



Cluster OpenMP プログラム 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

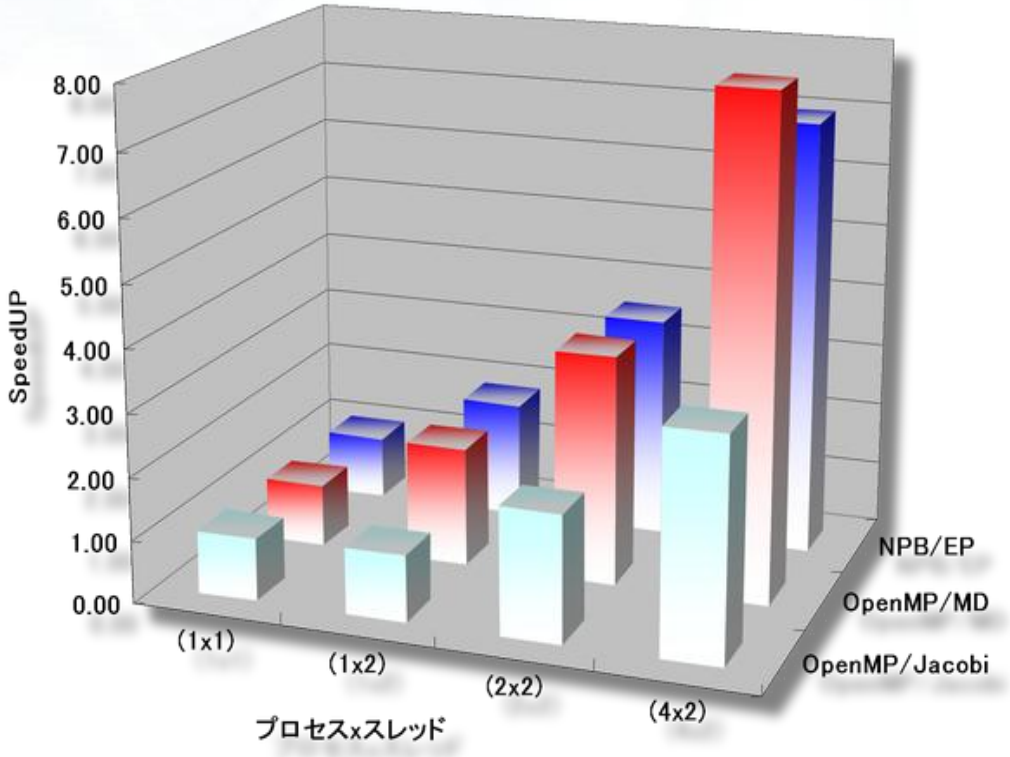
並列実行処理環境の設定

```
$ 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
```

Cluster OpenMP プログラム スケーラビリティサンプル

ベンチマークシステム
- NEXXUS 4820-PT
- 2.66GHz/1066MHz FSB/16GB Memory/InfiniBand

プログラムサンプル
- NAS Parallel Benchmark / EP ベンチマーク
- OpenMPサンプルプログラム（分子動力学サンプル、nparts=10000で実行）
- OpenMPサンプル(Jacobi法サンプル、5000x5000）



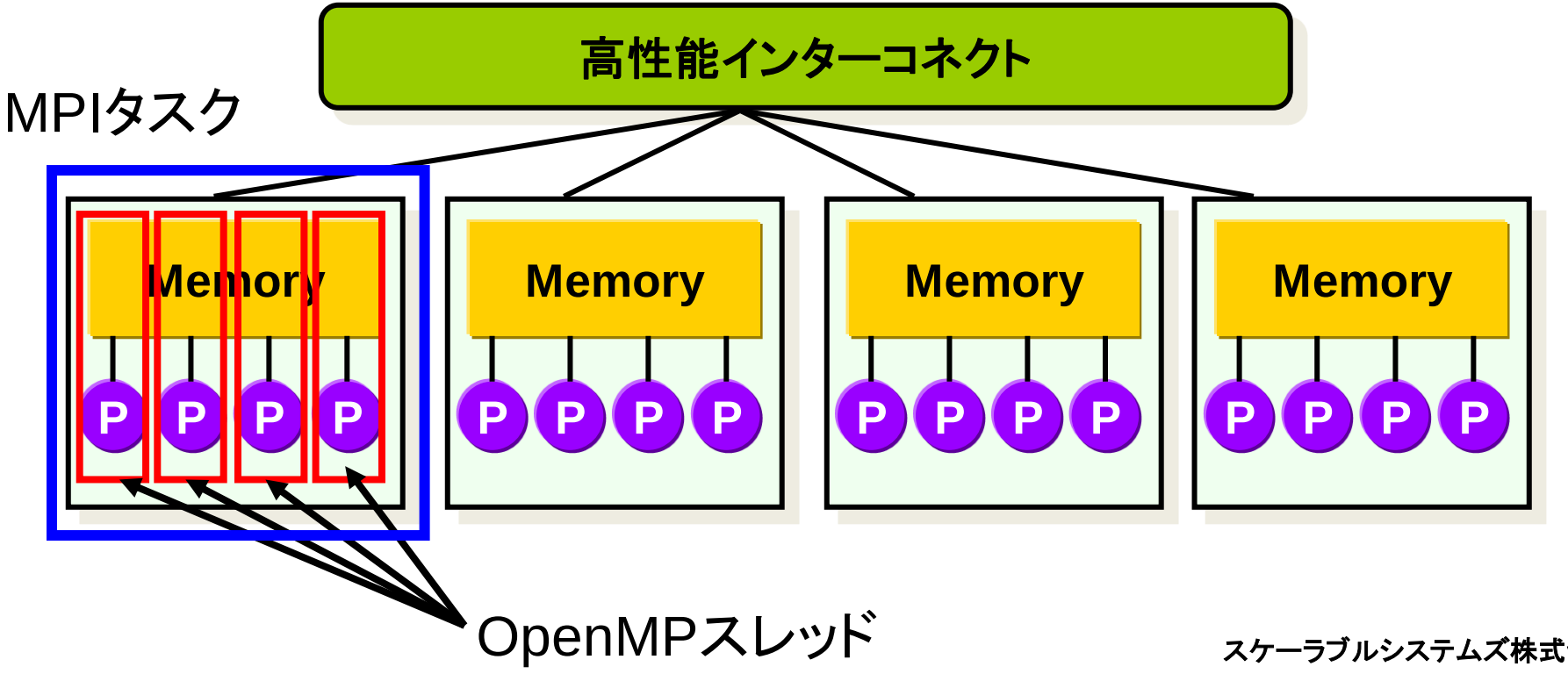
シングルAPIでの並列処理

MPI
OpenMP

Cluster OpenMPは、ノード内（SMP)とノード間で同一の並列APIでのプログラミングを可能とします。

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

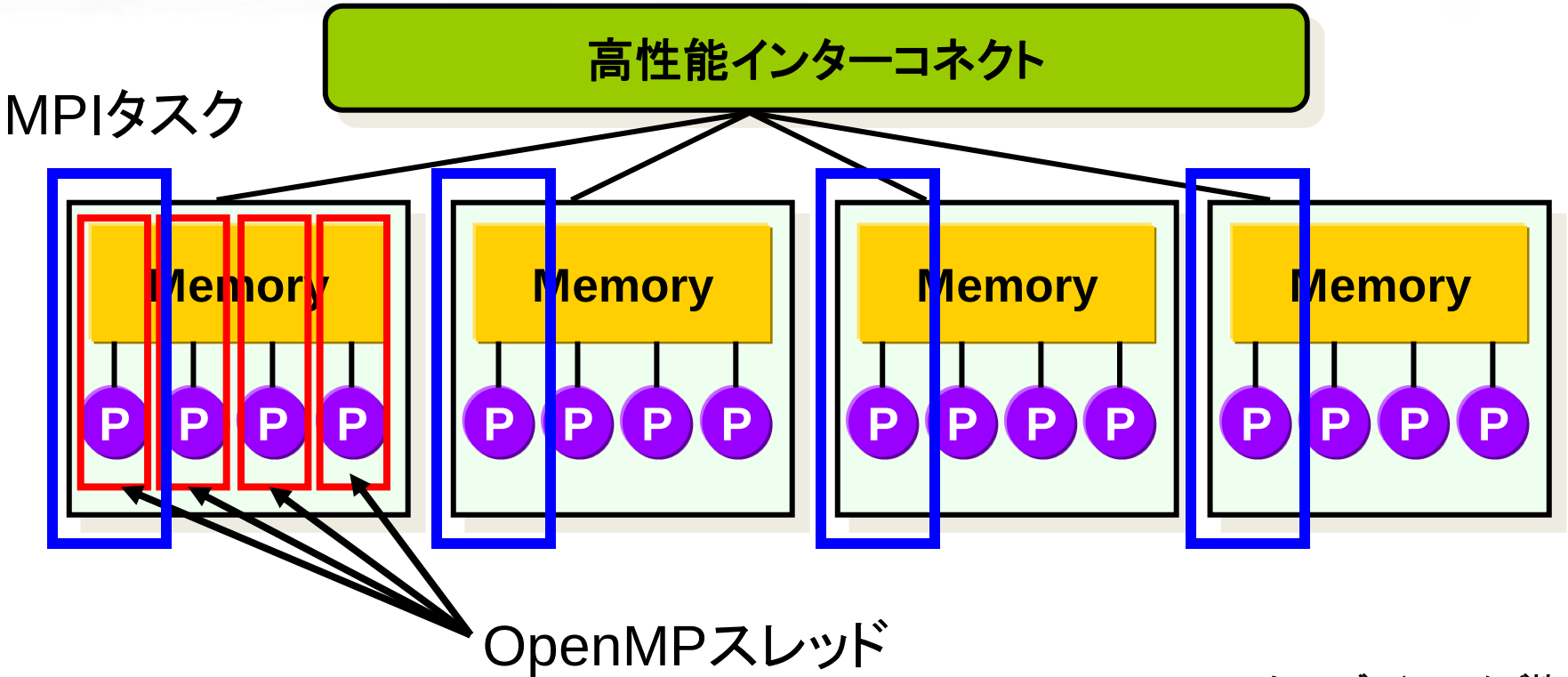
```
include ‘mpif.h’
program hybsimp


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
```

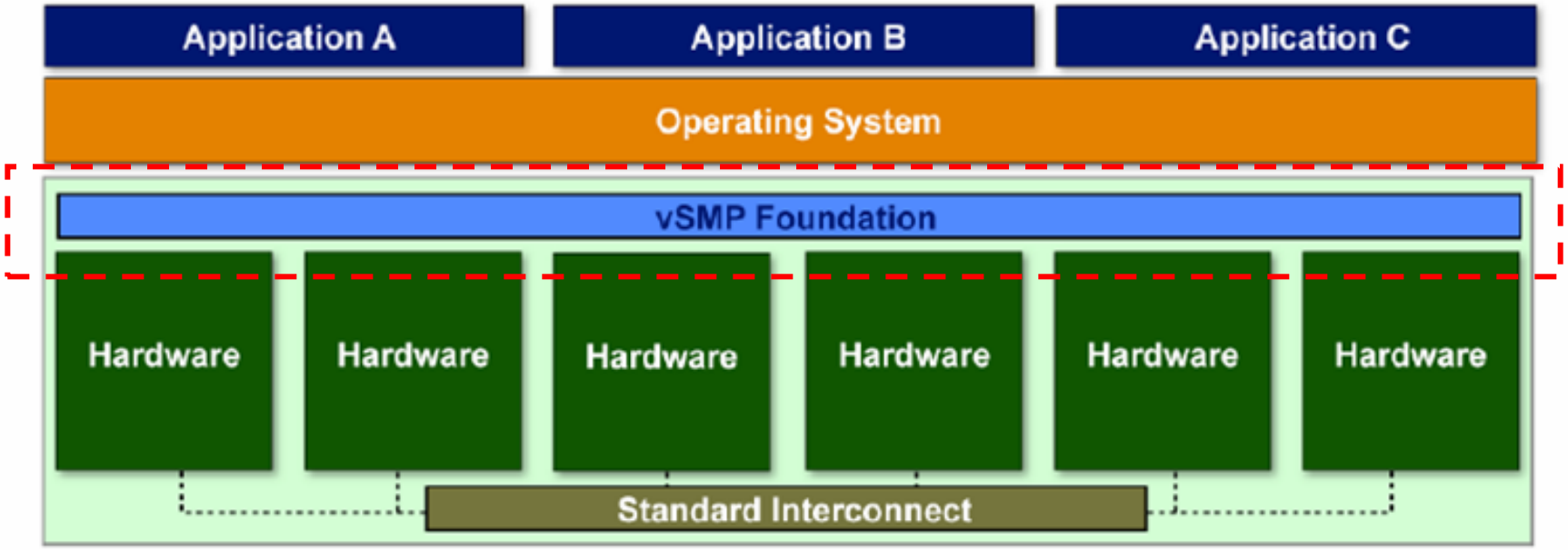
C/C++

```
#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

}
```

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



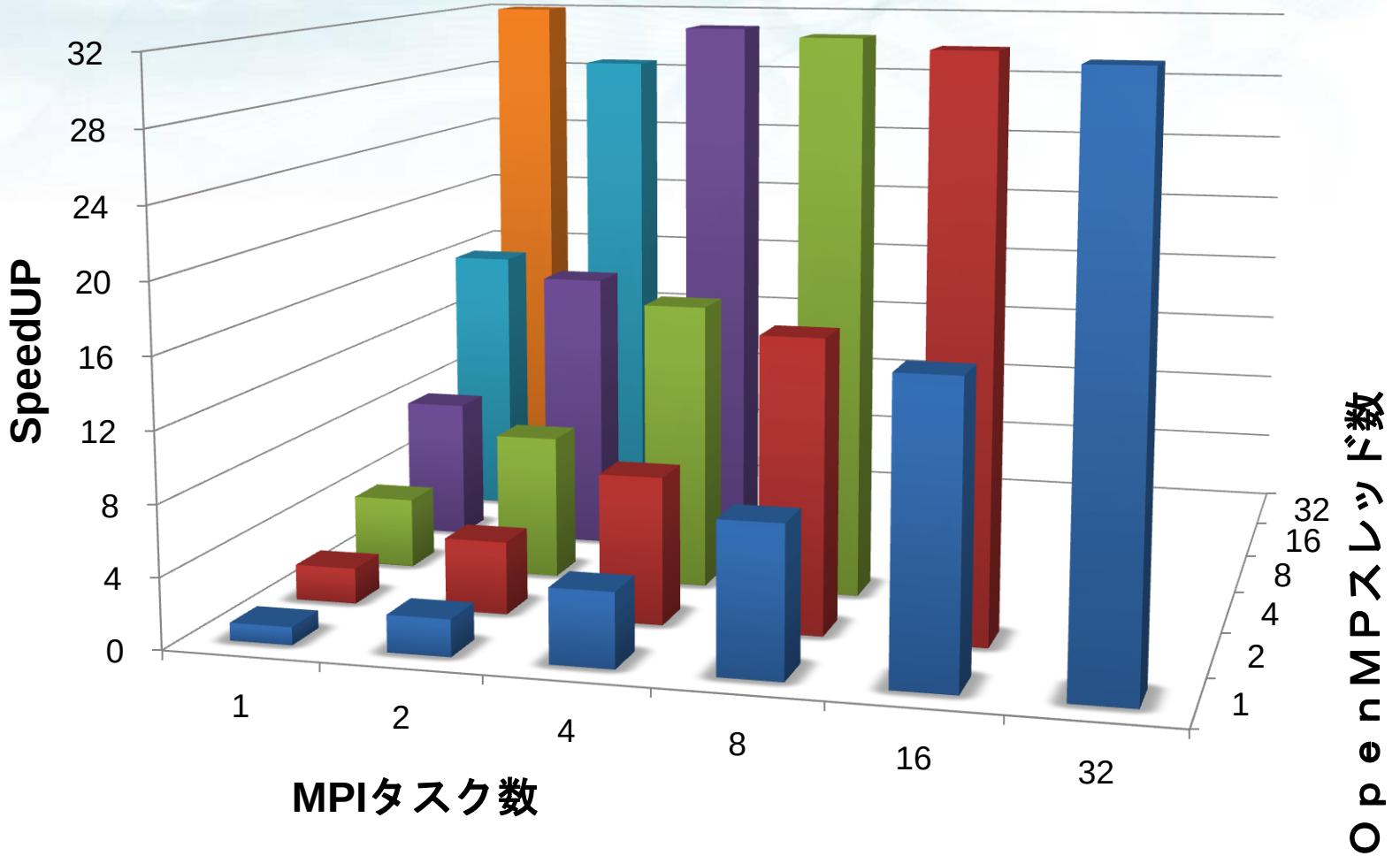
Hardwareは一般のx86チップセットと標準インターコネクトでシステムの構築が可能

vSMP Foundation でのシステムのSMP拡張を実現

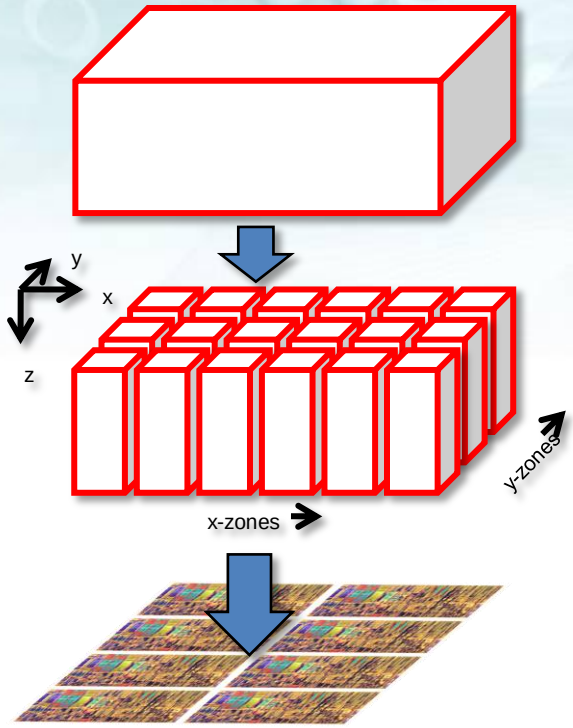
OpenMP/MPI/ハイブリッド

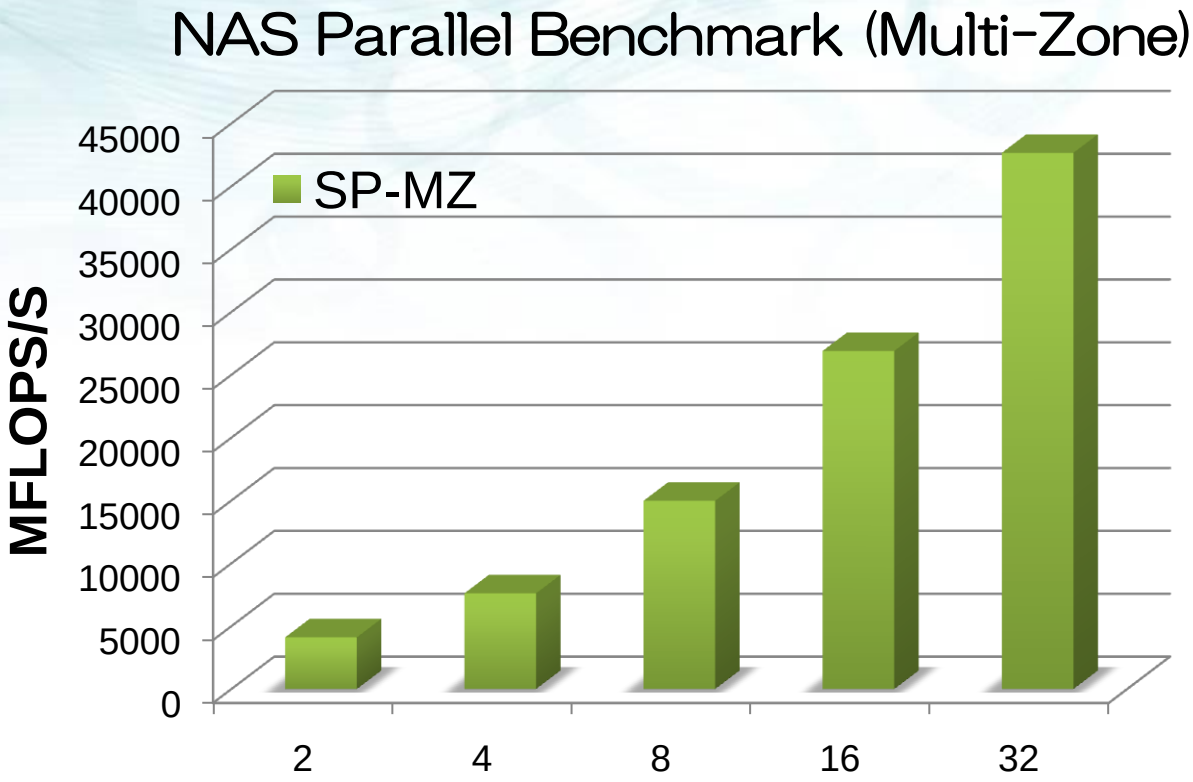
Hybrid OpenMP MPI Benchmarkproject (″homb″)

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.



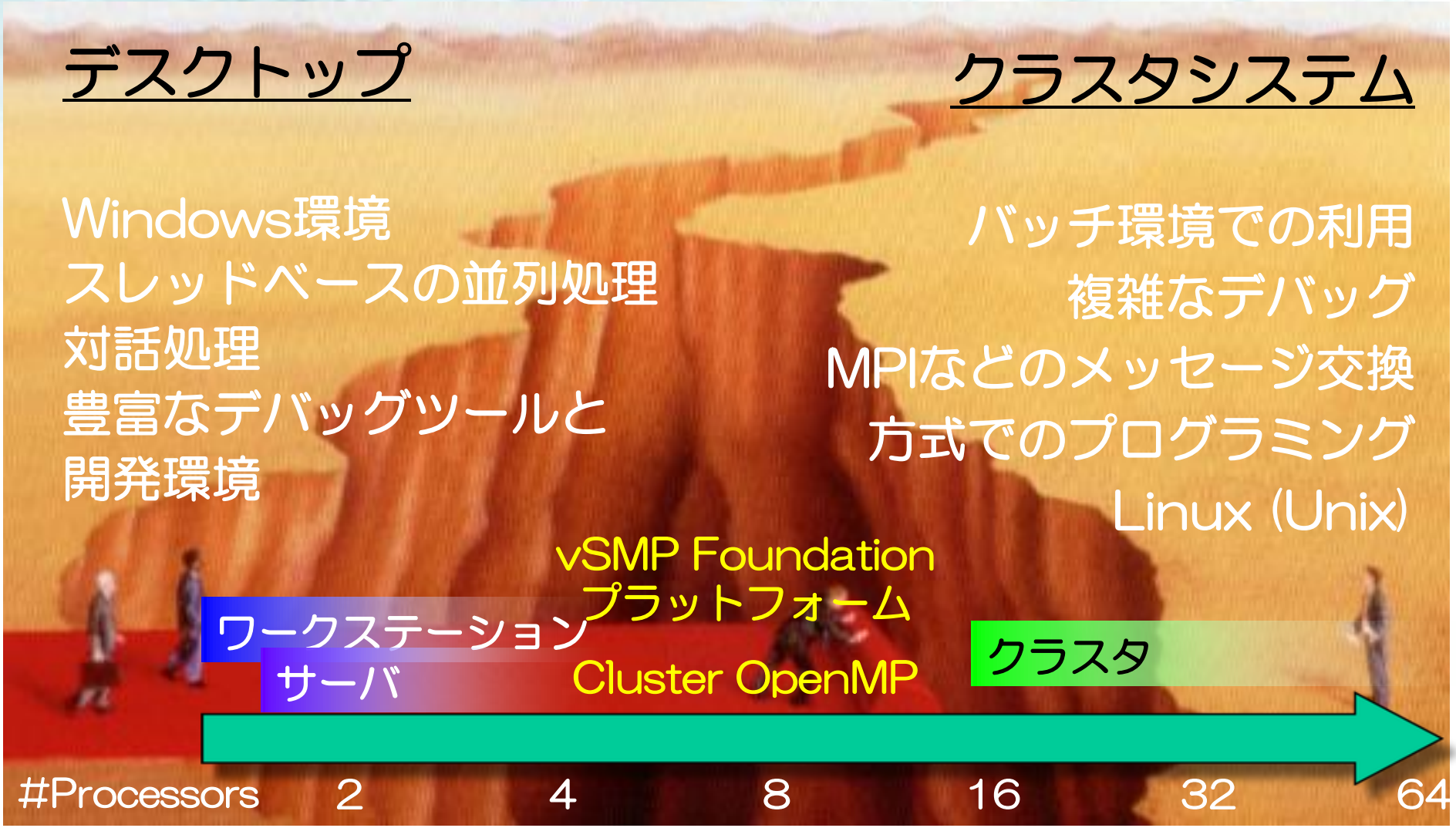
OpenMPベンチマーク

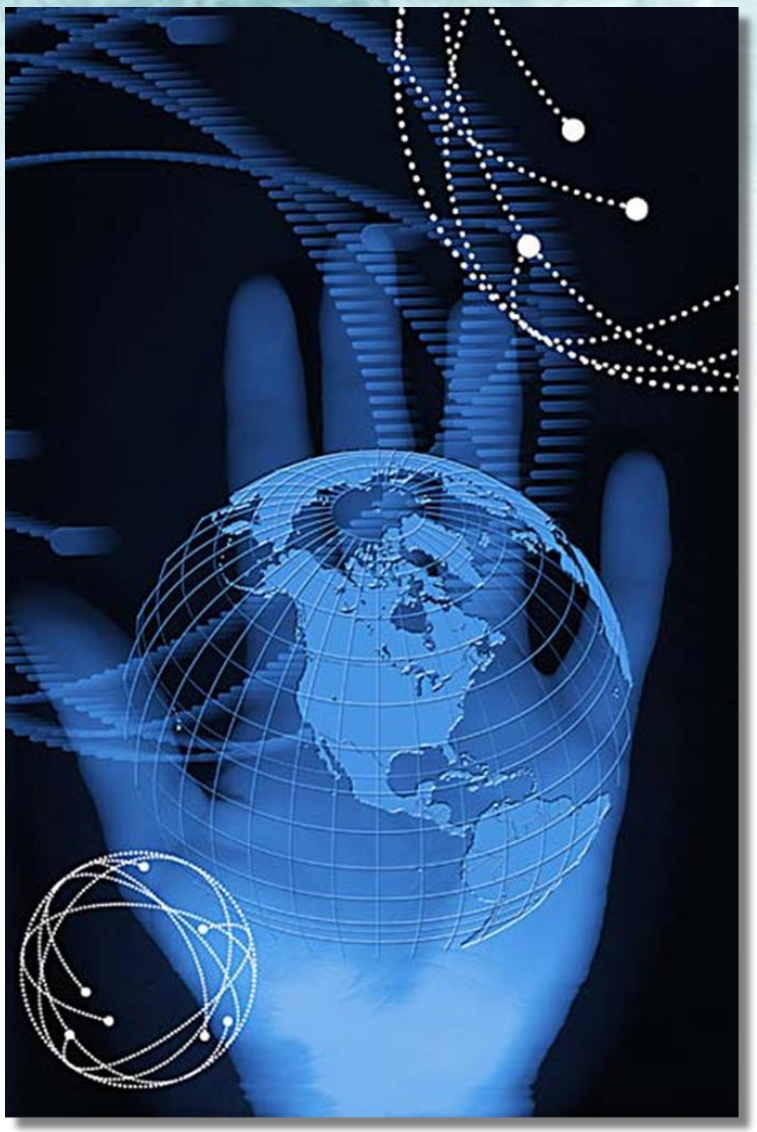




著名な公開ベンチマークツールである NAS Parallel Benchmark (NPB) の一つであるNPB-MZ (NPB Multi-Zone)はより粒度の大きな並列化の提供を行っています。NPB-MZでは、ハイブリッド型の並列処理やネストしたOpenMPのテストが可能です。ここでの結果は、OpenMPだけでの並列処理の性能を評価しています。
Xeon 5550 (2.66GHz) vSMP Foundation

ソフトウエアのギャップの解決





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

10/16/2009

スケーラブルシステムズ株式会社