理化学研究所
スーパーコンピュータシステムRICC
について

理化学研究所
情報基盤センター
重谷　隆之

これまでの理研スーパーコンピュータシステム

- ～1994年2月：メインフレーム（FUJITSU Mシリーズなど）

- 1994年2月～：ベクトル計算機（Fujitsu VPP500）
- 1999年2月～：ベクトル計算機（Fujitsu VPP700E）

- 2004年3月～：RSCC：PCクラスタ＋ベクトル計算機＋専用計算機の複合システム

- 2009年8月～：RICC：PCクラスタ＋専用計算機＋GPGPU

PCクラスタに関する理研での取り組み

2000年 デスクトップPC
CPU: Pentium III 450MHz × 1CPU × 8ノード
Interconnect : Fast Ethernet x 1 (100Mbps)

2001年 デスクトップPC
CPU: Pentium4 1.5GHz × 1CPU × 8ノード
Interconnect : Fast Ethernet x 1 (100Mbps)

2001年 ラックマウント4U
CPU: Pentium 4 1.7GHz × 1CPU × 64ノード
Interconnect : Myrinet 2000
Peak :217.6 GFLOPS

大規模PCクラスタ

2004年 RSCC
CPU: Xeon 3.06GHz × 2CPU × 1024ノード
Interconnect : InfiniBand
Peak：12.4 TFLOPS

2001年 Score III
CPU: Pentium III 933MHz × 2CPU × 512ノード
Interconnect : Myrinet 2000 (2.0Gbps)
Peak：955.4 GFLOPS

2004年 AISTスーパークラスタ
CPU: Opteron 2.0GHz × 2CPU, Itanium2 1.3GHz × 4CPU、Xeon 3.06GHz × 2CPU
Interconnect : Myrinet、GbE
Peak：14.6TFLOPS

2009年6月まで運用していたRSCC

RIKEN Super Combined Cluster

【システム構成】
スカラ＋ベクトル＋専用機
の複合システム

【スカラ部】
12.4テラフロップス

【ベクトル部】
0.28テラフロップス

入出力機器

【システム間接続】
1ギガビット毎秒のネット
ワークを使いグリッド接続

MD-GRAPE3：64テラフロップス追加(07年)

産業技術大賞
文部科学大臣賞・受賞
2005年4月

【特徴】

- 世界初のスカラ+ベクトル+専用機複合システム
  - –計算機センターでPCクラスタを採用
- 日本で初めてグリッド技術を全面的に採用した計算機センター
  - –利用者に利用計算機を意識させない
- 世界最大規模、日本で最速のPCクラスタ
  - –Top500リスト（2004年6月）第7位
  - –高性能で低コスト

次世代スーパーコンピュータ開発のテストベッドとして使用

RSCCの狙い

1. スーパーコンピュータの既成概念に縛られず、コスト性能比の良い計算機をメインの計算機に
2. 新たな利用者の獲得
   - 実験データ処理、バイオインフォマティックスに適したシステムとし、理研内の新たな利用者を取り込む
   - Grid Computing、Web技術を用いて、それまで計算機センターのシステムに不慣れな研究者でも簡単に使えるシステムを構築
   - フリーソフトが多数利用できる計算機システムとすること
3. 従来の利用者も利用できる環境は維持

RSCCシステム開発と主な成果

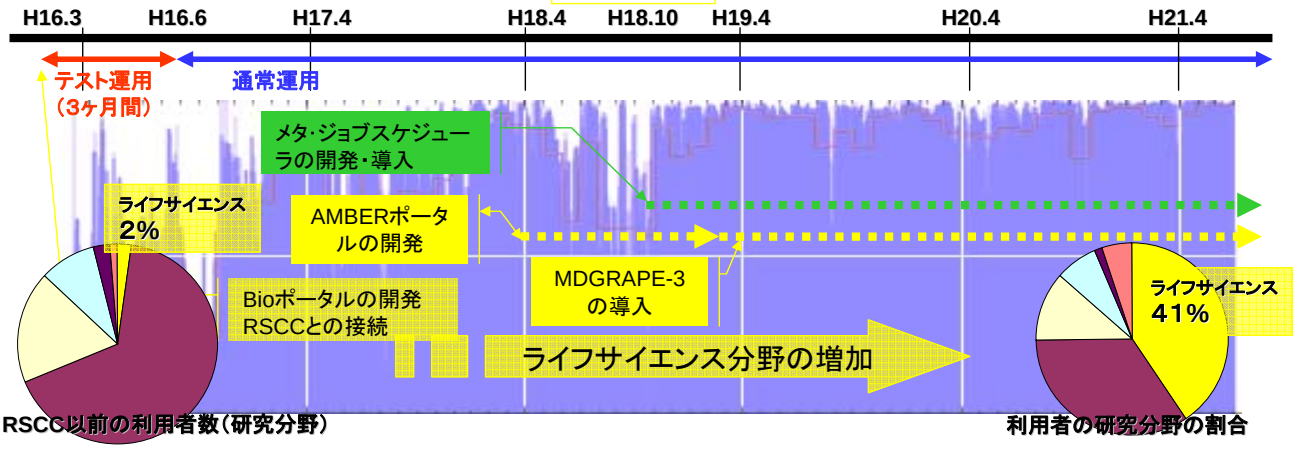
LINPACK性能で
世界7位

産業技術大賞
文部科学大臣賞・受賞

導入

主なプレス発表

- ★電子の磁石の強さを1兆分の1の精度まで計算
- ★汚い”物質中の電子が持つ美しい対称性『共形不変性』を世界で初めて実証 - 不規則系の臨界現象における理論手法の構築の第一歩 -
- ★素粒子の世界の真空エネルギーをコンピュータで計算
- ★金属表面で起こる分子の選択的分解反応と表面拡散運動の可視化に成功



2009年8月に運用開始した RIKEN Integrated Cluster of Clusters (RICC)

RICCの狙い

1. 次世代スーパーコンピュータに向けたアプリケーション開発環境の整備
   - 大規模並列に対応するために、8000コア超の大規模並列ジョブ実行を推進
   - システム・ソフトウェア（ジョブ・スケジューラ）の機能強化
2. 新しいトレンド：GPGPUアクセラレータへの挑戦
   - GPGPUアクセラレータを導入、利用を推進

RICCの概要

【システム構成】
超並列PCクラスタ＋GPUクラスタ＋専用機クラスタ＋大容量メモリ計算機
を単一の高速ネットワークで接続したクラスタ・オブ・クラスタ

演算性能：8.5倍
メモリI/O性能：2.5倍

実験データ

研究者

【超並列PCクラスタ】1024Nodes（8192core）
ノード性能：93.0GFLOPS, 12GB(mem),
500GB(hdd)，DDR IB×1

＊導入時、最新のCPUを採用した日本初の大規模PCクラスタ（8192 コア）

【多目的PCクラスタ】100Nodes(800core)
9.3テラフロップス
＋【GPGPUアクセラレータ】93.3TFLOPS

GPGPUの利用を容易にするためのビジュアル・プログラミング環境を日本IBMと共同で開発中

2009年導入時点のTOP500リストで世界40位、日本で3位、PCクラスタ・システムでは日本最速

容量10倍
I/O性能12倍

テープアーカイブ装置
【HPSS】(4PB)

容量27倍
I/O性能10倍

磁気ディスク装置
(550TB)

【専用機クラスタ】3TFLOPS
＋【MDGRAPE-3】64テラフロップス
＊理研で開発した分子動力学専用計算機を接続

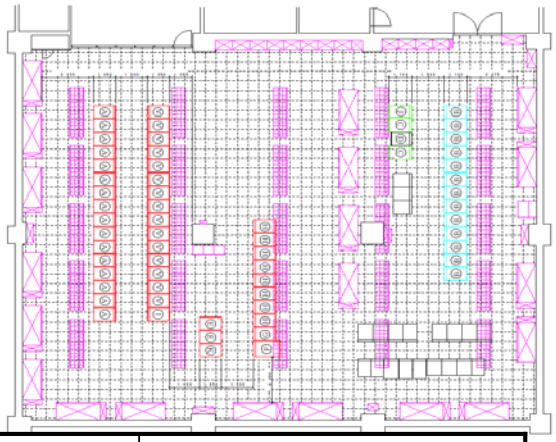
【大容量メモリ計算機】
0.24テラフロップス、512GBメモリ
＊1プロセスで500GB以上のメモリを利用可能

メモリ容量2倍

RICCの主な特徴

- 8000core超を同時に使えるインターコネクト（InfiniBand）構成
  - RSCCでは、サブクラスタごと、各サブクラスタ間はGbEで接続
- メタ・ジョブ・スケジューラの機能の充実
  - フェアシェア機能、バックフィル機能
  - マルチ・コアシステムでの効率的なジョブ管理・リソース管理
  - ネットワーク・トポロジと利用方針を踏まえたノード・アロケーション管理
  - 数万の単一CPU利用ジョブのスケジューリングに対応

システム緒元



	RSCC	RICC
理論性能	12.6TFLOPS	107.7TFLOPS + 64TFLOPS (MDGRPAE) + 93.3TFLOPS (GPU;SP)
設置面積（テープアーカイブ装置＆保守スペース除く）	約40㎡	約47㎡
重量	約24t	約40t
消費電力（ピーク）	約660kVA	約850kVA
発熱量	約550Mcal/h	約710Mcal/h

マシン室写真

超並列PCクラスタ

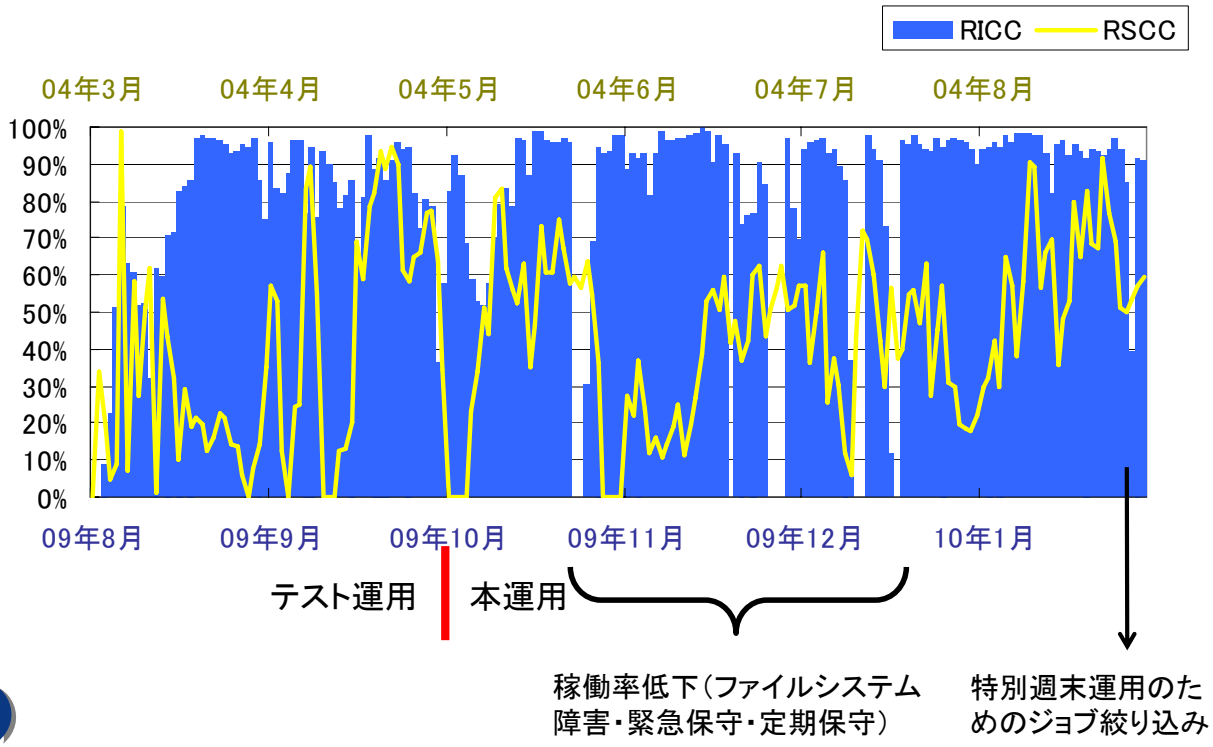
増設した空調機

多目的PCクラスタ（GPGPU搭載）

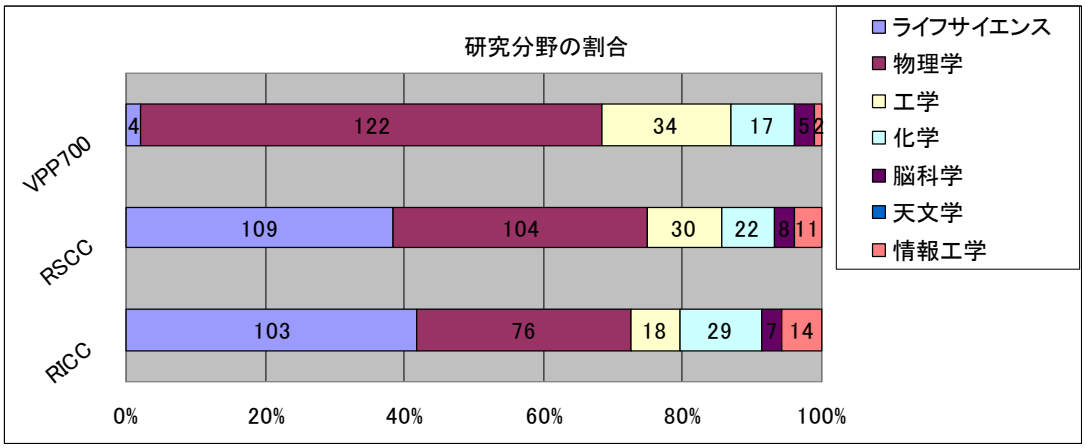
MDGRAPE-3クラスタ＆
大容量メモリ計算機

RICC運用状況

超並列PCクラスタの利用率

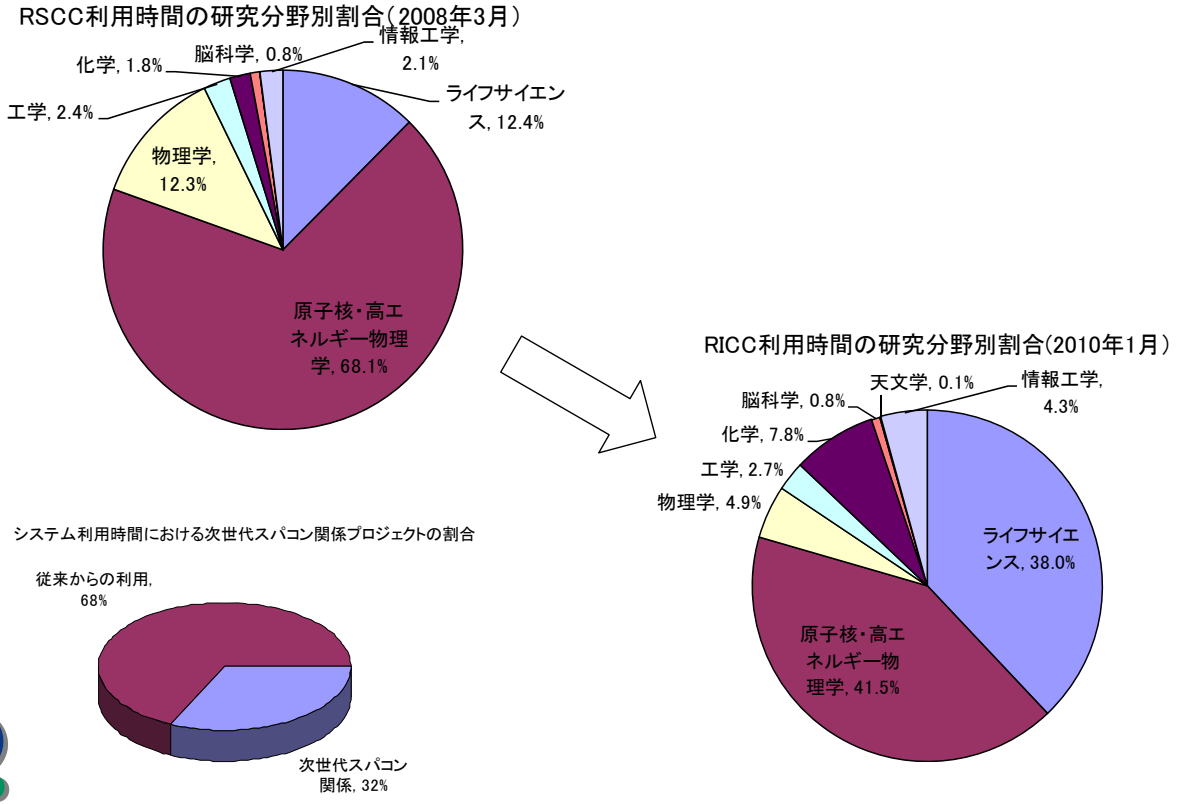


研究分野ごとの利用者数
VPP700/RSCCとの比較

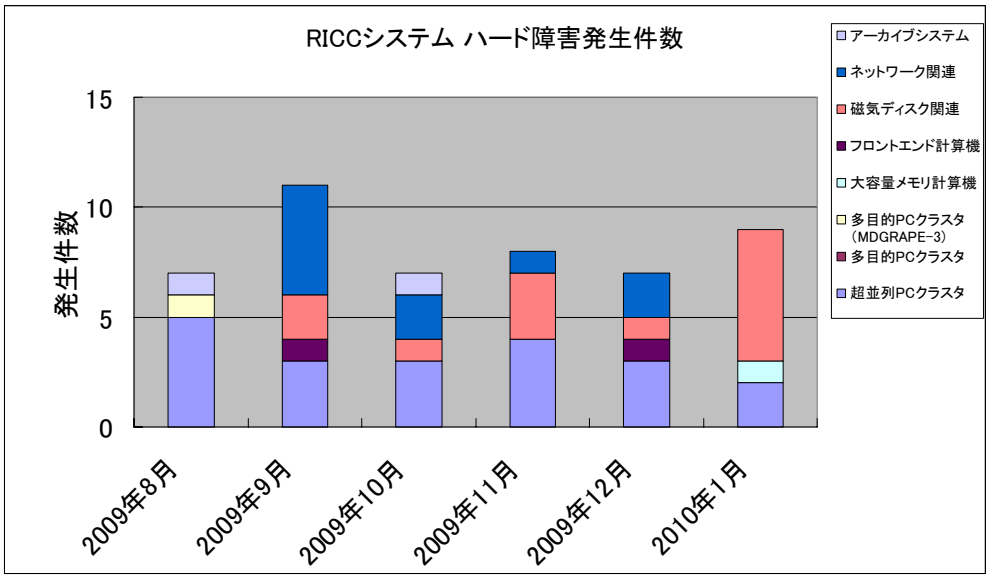


- VPP700（2004年2月末）、RSCC(2009年6月末）とRICC（2010年1月末）の登録ユーザーの研究分野による分類比較
- VPP700と比べると、研究分野では、ライフサイエンスが大幅に増大

研究分野ごとの利用時間割合



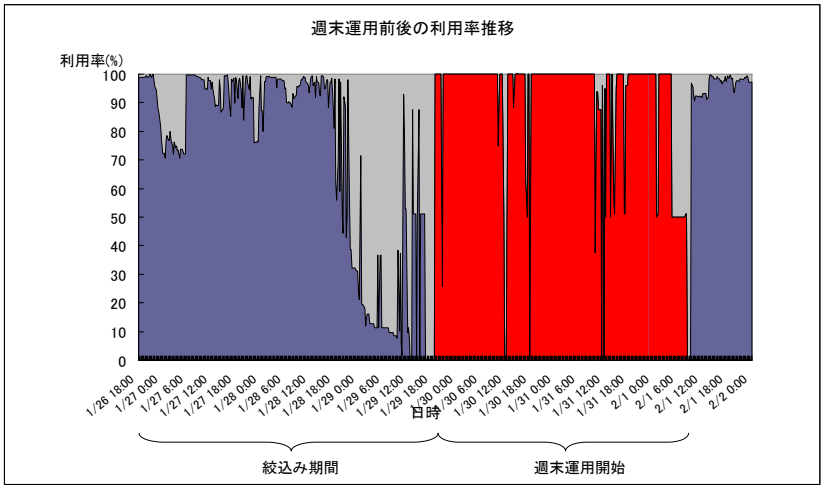
RICCハードウェア故障



- 予防交換を除いたクラスタの平均故障率は約3.5台/月
- RSCC：5年間の平均故障率は約3.4台/月

大規模並列ジョブのための週末運用

- 高い稼働率のため通常運用中に大規模並列ジョブの実行は困難
- 大規模並列ジョブをある程度まとめて、特別週末運用時間を設けて対応
- 2010年1月から開始（1ヶ月に2週末）
- 予定してジョブを絞り込むため、一般のジョブは実行時間を短く設定



GPGPUの利用促進に向けて

GPUプログラムの問題

LU分解のオリジナル・プログラム（一部）

```
void kernelD(
        const Matrix<T, Z, C>& blockB,
        const Matrix<T, R, Z>& blockC,
        const Matrix<T, R, C>& blockD,
        Matrix<T, R, C>& result)
{
        struct timeval tvs, tve;
        std::stringstream ss;
        int i, j, k;
        gettimeofday(&tvs,NULL);
        ss << tvs.tv_sec << "." << tvs.tv_usec << " kernelD" << R << " start."  << std::endl;
        std::cerr << ss.str();
        ss.str("");

        /* To make the code simpler, input matrix is copied to the output one first */
        for(i = 0; i < R; i++) // row
                for(j = 0; j < C; j++) // column
                        result.elementAt(i, j) = blockD.elementAt(i, j);

        /* Main loop of submatrix calculation */
        for (i = 0; i < R; i++) // row
                for (k = 0; k < Z; k++) // column or row
                        for (j = 0; j < C; j++) // column
                                result.elementAt(i, j) +=
        blockB.elementAt(k, j) * blockC.elementAt(i, k);

        gettimeofday(&tve,NULL);
        ss << tve.tv_sec << "." << tve.tv_usec << " kernelD" << R << " finish."  << std::endl;
        tve.tv_usec -= tvs.tv_usec;
        tve.tv_sec -= tvs.tv_sec;
        if( tve.tv_usec < 0 ){
                tve.tv_usec += 1000000;
                tve.tv_sec--;
        }
        ss << tve.tv_sec << "." << tve.tv_usec << " kernelD" << R << " used."  << std::endl;
        std::cerr << ss.str();
        ss.str("");
}
```

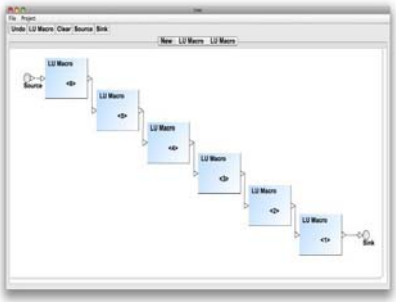
GPGPU版プログラム（一部）

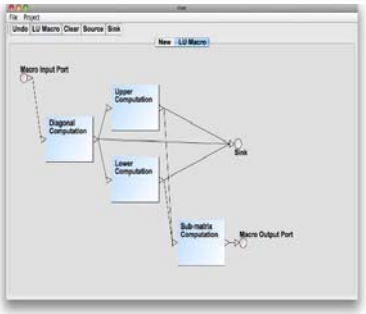
```
void kernelD(
    Matrix<float,MATRIX_SIZE,MATRIX_SIZE>* blockD,
    Matrix<float,MATRIX_SIZE,MATRIX_SIZE>* blockB,
    Matrix<float,MATRIX_SIZE,MATRIX_SIZE>* blockC,
    Matrix<float,MATRIX_SIZE,MATRIX_SIZE>* result);

extern "C"
void* udopLU_D(void* parm)
{
    uspade_udop_parm_t* uparm = (uspade_udop_parm_t*)parm;
    std::string blockDParm = "127.0.0.1:10003";
    std::string blockBParm = "127.0.0.1:10001";
    std::string blockCParm = "127.0.0.1:10002";
    std::string resultParm = "127.0.0.1:10004";

    for (std::map<std::string, std::string>::const_iterator it = uparm->parms.begin(); it != uparm->parms.end(); it++)
    {
        size_t pos;
        while ( (pos = blockDParm.find(it->first)) != std::string::npos ) blockDParm.replace(pos, it->first.length(), it->second);
        while ( (pos = blockBParm.find(it->first)) != std::string::npos ) blockBParm.replace(pos, it->first.length(), it->second);
        while ( (pos = blockCParm.find(it->first)) != std::string::npos ) blockCParm.replace(pos, it->first.length(), it->second);
        while ( (pos = resultParm.find(it->first)) != std::string::npos ) resultParm.replace(pos, it->first.length(), it->second);
    }

    InSocketPort<Matrix<float,MATRIX_SIZE,MATRIX_SIZE> > blockDPort(blockDParm);
    InSocketPort<Matrix<float,MATRIX_SIZE,MATRIX_SIZE> > blockBPort(blockBParm);
    InSocketPort<Matrix<float,MATRIX_SIZE,MATRIX_SIZE> > blockCPort(blockCParm);
    OutSocketPort<Matrix<float,MATRIX_SIZE,MATRIX_SIZE> > resultPort(resultParm);

    Matrix<float,MATRIX_SIZE,MATRIX_SIZE> blockD;
    Matrix<float,MATRIX_SIZE,MATRIX_SIZE> blockB;
    Matrix<float,MATRIX_SIZE,MATRIX_SIZE> blockC;
    Matrix<float,MATRIX_SIZE,MATRIX_SIZE> result;

    while ( uparm->active )
    {
        if ( uparm->active ) blockDPort.receive(blockD);
        if ( uparm->active ) blockBPort.receive(blockB);
        if ( uparm->active ) blockCPort.receive(blockC);

        struct timeval tv_st, tv_ed;
        gettimeofday(&tv_st, NULL);

        if ( uparm->active ) kernelD(
                &blockD,
                &blockB,
                &blockC,
                &result);

        gettimeofday(&tv_ed, NULL);
        printf("kernel fired!! (at %f in msec, %f [msec] to process kernel)¥n",
                (double)tv_ed.tv_sec * 1000 + (double)tv_ed.tv_usec / 1000,
                (double)(tv_ed.tv_sec - tv_st.tv_sec - 1) * 1000 + (double)(1000000 + tv_ed.tv_usec - tv_st.tv_usec) / 1000);

        if ( uparm->active ) resultPort.send(result);
    }

    return NULL;
}
```

GPGPUアプリケーション開発環境

RIVER (Riken IBM Visual Programming EnviRonment)

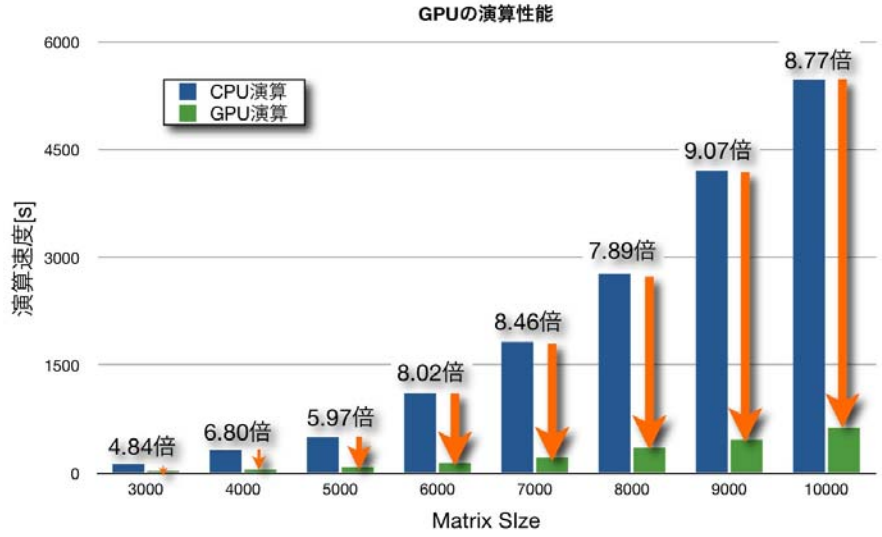
- GPGPUは高速だが、その性能を引き出すには高度なプログラムのスキルが必要
- だれでも使えるように初心者向けの、ビジュアル・プログラミング環境を日本IBMと共同で開発中
- 部品ライブラリの中の部品を組み合わせるだけで、プログラミングが可能
- ノード並列もサポート予定





連立一次方程式の前処理プロセス：LU分解の例

LU分解の測定結果（10並列）



RIVERの現状と今後

- ポイントは、部品ライブラリの整備
  - 部品が優秀なら、そこそこの性能
- 現在は理研内のアプリに応用してテスト
  - 部品ライブラリーを整備していく予定
- 将来は、RIVERと部品ライブラリーをフリーフェアとして配布

RICCの現状と今後

- 8000並列超の並列性能のテスト
- 運用開始直後から高い利用率
  - 特別週末運用の実施
  - GPGPUの利用促進をして、CPUからGPUへ誘導
    - RIVERの開発
    - 応用分野を開拓する必要

- 今後はスペースよりも電力、空調能力が問題に
  - PCクラスタ＋アクセラレータは電力性能比向上の1つの解
  - ノードやラック単位だけでなく、コンピュータルーム全体として、効率の良い冷却方式の検討が必要