計算機アーキテクチャ特論
(Advanced Computer Architectures)

12. メニーコアプロセッサ，並列プログラミング

吉瀬 謙二 計算工学専攻
kise _at_ cs.titech.ac.jp www.arch.cs.titech.ac.jp

1. 分岐予測の実装と評価

- **gshare分岐予測**を実装し，その予測ミス率を測定せよ．また，bimodal分岐予測との予測精度(20本のベンチマークのミス率の算術平均)の比較を示せ．
  - ハードウェア量を 2KB，4KB，8KB，16KB，32KB，64KBとしてグラフを描け．
- **gshare分岐予測**に工夫を施し（あるいは，異なる方式の予測を実装し），予測ミス率を測定せよ．
  - ハードウェア量を 2KB，4KB，8KB，16KB，32KB，64KBとしてグラフを描け．
  - **予測ミス率が低い（性能が高い）と高得点．**
- 提出済みの場合，この問題を解く必要はない．

2. マルチコアプロセッサ(並列)プログラミング

(2-1) プロセッサシミュレータSimMcを利用して，与えられる行列積のプログラム(test64) を4個のコア用に並列化せよ．
4個のコアを用いて，2倍以上の高速化を達成すること．
コンパイラの最適化オプションを利用しない (-O0を利用する) こと．
ソースコード及び性能向上率を示せ．また，この課題に要した時間を示すこと．

(2-2) (2-1)で作成したプログラムを(必要であれば)修正して，コアの数(1,2,4,8,16)と性能向上率との関係をグラフに示せ．また，この課題に要した時間を示すこと．
ここでも，コンパイラの最適化オプションを利用しない (-O0を利用する) ．
オリジナル・プログラムの性能を1として，グラフを描くこと．

(2-3) コンパイラの最適化オプションをO3として，コアの数(1,2,4,8,16)と性能向上率との関係をグラフに示せ．
オリジナル・プログラム (O3) の性能を1として，グラフを描くこと．
また，最適化オプションの影響を議論せよ．この課題に要した時間を示すこと．

2. マルチコアプロセッサ(並列)プログラミング

(2-4) コアの数を16に固定して，作成したプログラムに最適化(工夫)を施せ．最適化オプションをO3，オリジナル・プログラムの性能を1として，並列化による速度向上率を示せ．また，用いた最適化とその効果を説明せよ．

(2-5)
・プロセッサシミュレータSimMcについての感想をまとめよ．
どこで苦労したか？
どの程度の時間が必要となったか？
期待する改良点．

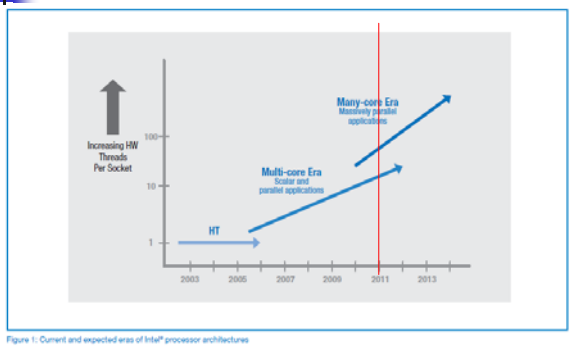
3. 計算機システムの展望

- 10年後の計算機システム(パーソナルコンピュータ)はどのような構成になっているだろうか？
  計算機アーキテクチャの視点から議論せよ．
- また，そのような計算機システムを活用するために解決すべき課題(研究テーマ)について考察せよ．

レポート 提出方法

- 2011年2月9日(水) 午後5時までに電子メール(PDFファイルで添付)にて提出
- kise at cs.titech.ac.jp

マルチコア（2個～数10個）からメニーコアへ



Platform 2015: Intel® Processor and Platform Evolution for the Next Decade

マルチコア（2個～数10個）からメニーコアへ

Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction, MICRO-36

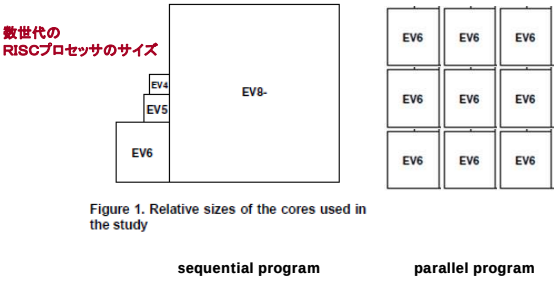
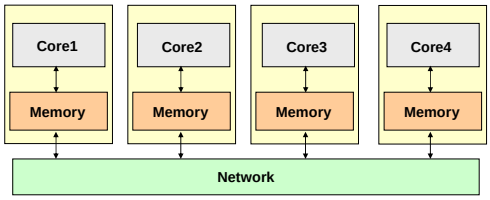


Figure 1. Relative sizes of the cores used in the study

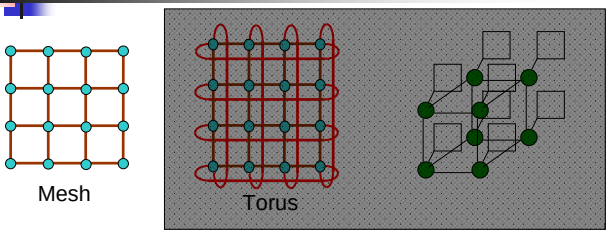
sequential program **parallel program**

ネットワーク結合のマルチコアプロセッサ



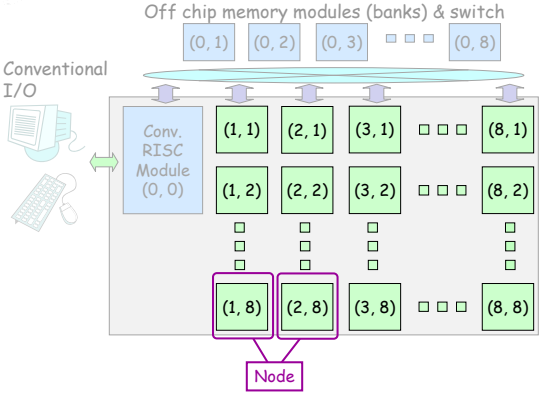
メニーコアプロセッサシミュレータ SimMc

2D and 3D Mesh/Torus Network



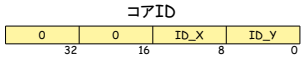
- N processors, N switches, 2, 3, 4 (2D torus) or 6 (3D torus) links/switch, 4N/2 links or 6N/2 links
- N simultaneous transfers
  - NB = link bandwidth * 4N or link bandwidth * 6N
  - BB = link bandwidth * 2 $N^{1/2}$ or link bandwidth * 2 $N^{2/3}$

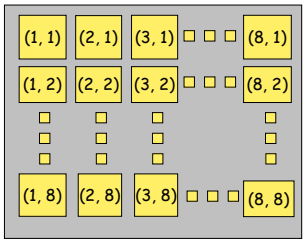
アーキテクチャモデル



SimMc: Many-Core and NoC Simulator

- 8ビットの整数 x, y を用いて，(x, y) の座標によりコアを指定する．x, yは0～255 の値をとる．ただし，x = 0 及び y = 0 は特別なユニットを表現するために予約する．y = 0 も使わない．
- Core ID は x, y の順序の連結 により生成される16ビットで表現する．

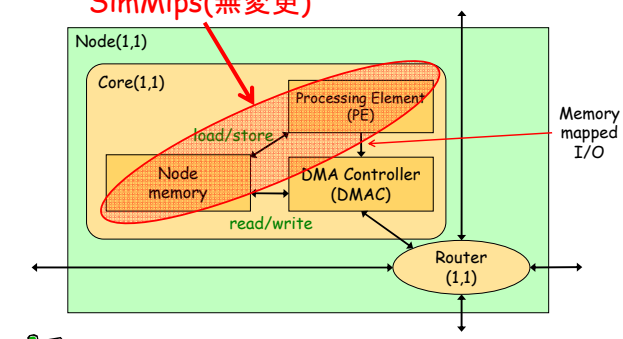




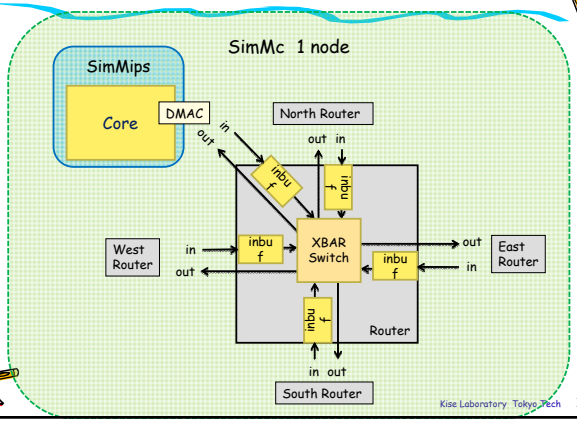
ネットワークアーキテクチャ

- トポロジ
  - メッシュ
- スイッチング
  - Warm hole, no virtual channel
- フロー制御
  - Xon / Xoff
- ルーティング
  - XY Dimension Order Routing

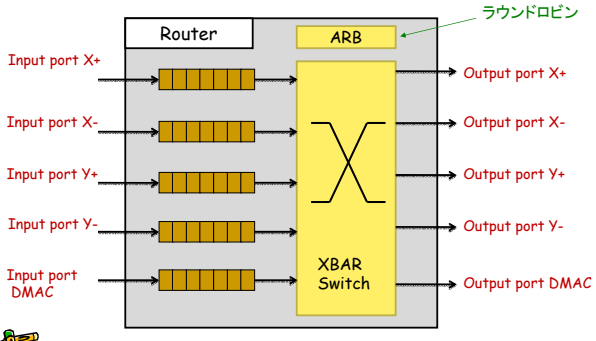
ノードの構成



SimMc

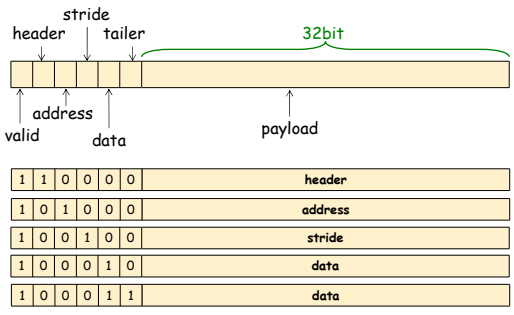


Router Architecture



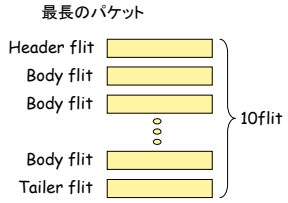
Packet および Flit の構成

- フリット(flit)は 38ビットの固定長とする

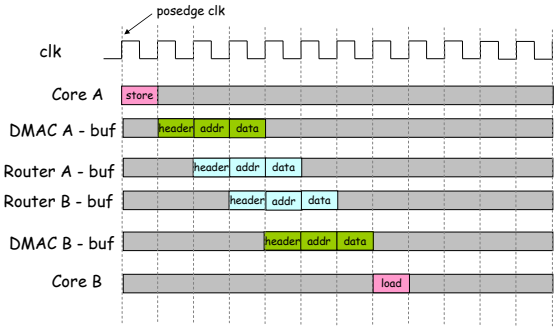


Packet および Flit の構成

- パケット(packet)は1つの header flit, 1～9個の address, stride, data flit であり, 最後のフリットは tailer のフラグを立てることによって構成される.
- パケットは最長で10flit である.
- フリット(flit)のサイズは 38ビットの固定長とする.

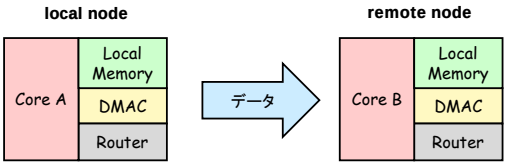


Core to Core の通信タイミング

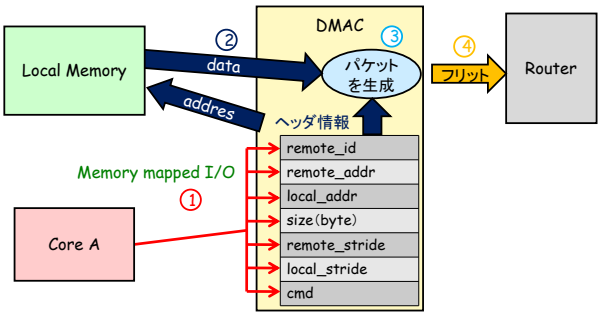


DMA 転送：MC_dma_put

- ローカルコアの保持するデータリモートコアのメモリに転送.
- 下の例は, コアAがMC_dma_putを呼び出し, コアBにデータを送る場合.



MC_dma_putの流れ - Local-Core ～ Router



Library: Multi-Core library **MC**lib

- int MC_init(int *id_x, int *id_y, int *rank_x, int *rank_y);
- void MC_finalize();
- void MC_dma_put(int dst_id, void *remote_addr, void *local_addr, size_t size, int remote_stride, int local_stride);
- void MC_dma_get(int get_id, int local_id, void *remote_addr, void *local_addr, size_t size, int remote_stride, int local_stride);
- int MC_printf(char *format, ...);
- void MC_puts(char* s);
- int MC_sprintf(char *buf, char *format, ...);
- int MC_sleep(int n);
- int MC_clock(unsigned int*);
- etc

サンプルプログラム

test10

kterm

```
/**************************************************************************/
/* Many-Core Architecture Research Project       Arch Lab. TOKYO TECH */
/**************************************************************************/
#include "MClib.h"

extern int cx, cy;

/**************************************************************************/
int main(int argc, char *args[])
{
    int id_x, id_y, rank_x, rank_y;
    MC_init(&id_x, &id_y, &rank_x, &rank_y);

    printf("$$ test10: I am core (%d,%d).\n", id_x, id_y);

    MC_finalize();
    return 0;
}

/**************************************************************************/
~
(END)
```


test22

kterm

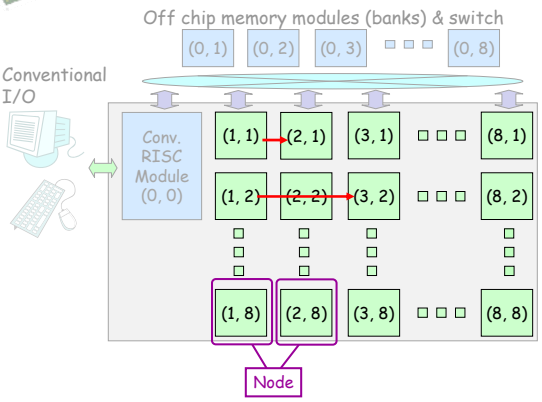
```
/**************************************************************************/
/* Many-Core Architecture Research Project       Arch Lab. TOKYO TECH */
/**************************************************************************/
#include "MClib.h"

/**************************************************************************/
int main(int argc, char *args[])
{
  int i;
  int id_x, id_y, rank_x, rank_y;
  MC_init(&id_x, &id_y, &rank_x, &rank_y);

  for(i = 0; i < 2; i++) {
    unsigned long long time;

    MC_clock(&time);

    printf("!! $$ test22: I am core (%d,%d) time: %d\n",
           id_x, id_y, (int)time);
  }

main.c
```


test31

kterm

```
/**************************************************************************/
/* Many-Core Architecture Research Project       Arch Lab. TOKYO TECH */
/**************************************************************************/
#include "MClib.h"
volatile int array[1024];

/**************************************************************************/
int main(int argc, char *args[])
{
    int i;
    int id_x, id_y, rank_x, rank_y;
    MC_init(&id_x, &id_y, &rank_x, &rank_y);

    for (i = 0; i < 1024; i++)
        array[i] = 0;

    if (id_x == rank_x && id_y == rank_y) {
        for (i = 0; i < 1024; i++)
            array[i] = 777;
        int size = 128;
        int stride = 4;
        int dst = 0;
        setidx_y(&dst, 1, 1);
        MC_dma_put(dst, &array, &array, size, stride, stride);
    }
    MC_sleep(5000);
    if (id_x == 1 && id_y == 1) {
        printf("array[0] %d\n", array[0]);
    }
    MC_finalize();
    return 0;
}

/**************************************************************************/
main.c (END)
```


通信遅延



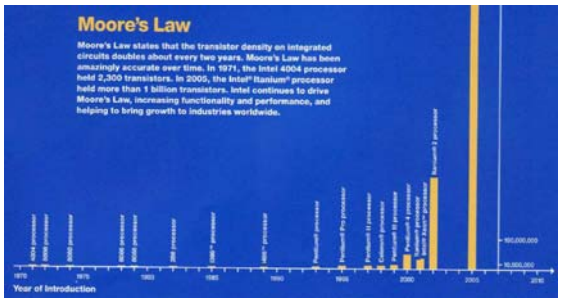
test64 (1/2)

```
/**************************************************************************/
void worker()
{
    slave_init();
    matrixproduct();
    slave_finalize();
}

/**************************************************************************/
int main(int argc, char *args[])
{
    int id_x, id_y, rank_x, rank_y;
    MC_init(&id_x, &id_y, &rank_x, &rank_y);

    if (id_x == 1 && id_y == 1) {
        printf("$$ I am %s¥n", MC_APP_VER);
        master();
    } else if (id_x == rank_x && id_y == rank_y) {
        worker();
    }

    MC_finalize();
    return 0;
}

/**************************************************************************/
(END)
```


test64 (2/2)

```
/**************************************************************************/
void matrixproduct()
{
    int i, j, k, sum;
    int row_buf[DIM];
    int col_buf[DIM];
    int work[DIM];
    for (i = 0; i < DIM; i++) {
        row_buf[i] = 0;
        col_buf[i] = 0;
        work[i] = 0;
    }
    for (i = 0; i < DIM; i++) {
        get_row(row_buf, i);
        for (j = 0; j < DIM; j++) {
            get_column(col_buf, j, STRD);
            sum = 0;
            for (k = 0; k < DIM; k++)
                sum += row_buf[k] * col_buf[k];
            work[j] = sum;
        }
        put_row(work, i);
    }
}

/**************************************************************************/
```

Moore's Law


Moore's Law
Moore's Law states that the transistor density on integrated circuits doubles about every two years. Moore's Law has been amazingly accurate over time. In 1971, the Intel 4004 processor held 2,300 transistors. In 2005, the Intel® Itanium® processor held more than 1 billion transistors. Intel continues to drive Moore's Law, increasing functionality and performance, and helping to bring growth to industries worldwide.
Year of Introduction