



PARRAY : 针对GPU集群的统一编程 工具简介

A Unified Programming Interface for GPU Clusters

陈一峯

Chen, Yifeng

北京大学众核软件研究小组

PKU Manycore Software Research Group



北京大学
PEKING UNIVERSITY

Portable Software for GPU Clusters?

- ◆ Nebulae: 1 GPU per node
- ◆ Mole-8.5: 6 GPUs per node
- ◆ PKU McClus: 2 GPUs per node, 80Gb/s IB





Time for Change?

- “If we buy a GPU cluster, how can we make use of it?”
- “My MPI legacy codes do not work with GPU.”
- “Upgrading them is like rewriting the whole thing.”
- “My code is optimized for this machine but perhaps not that one with 6 GPUs per node.”
- “I use MatLab, who will develop portable basic math libraries for me?”

Existing tools don't work.

Time to try something new?

Yes, we can!



北京大學
PEKING UNIVERSITY



Dilemma of Parallel Programming

- ◆ Approach A — **hiding** and abstraction
 - Programmers ignore low-level features of the architecture
 - OpenMP, TBB and Cilk.
- ◆ Approach B — **exposure** and distinction
 - Source program distinguishes different memory and communication
 - Pthread, CUDA, OpenCL, MPI and IB/verbs.
- ◆ Remarks
 - Over hiding puts too much burden on compiler.
 - Optimization only effective for specific apps (MapReduce).
 - General-purpose parallel computing requires control.
 - Over exposure puts too much burden on programmer.





Parallelization Arrays (PARRAY)

- ◆ Finding the underlying “mathematical structure” behind performance-important architectural features.
- ◆ Scaling up abstraction level with the mathematical structure.



Array with Nested Dimensions

Pinned Memory

```
$A[[3][2]][4]@PINNED(float)$
```

```
float* x;
```

```
$create(x):A$;
```

```
for (int i=0; i<$dim:A$; i++) x[$A[i]$]=i;
```

```
printf("number of rows = %d\n", $dim:A_0$);
```

```
printf("x[[1][1]][0] = %d\n", x[$A[[1][1]][0]$]);
```

cudaMallocHost()

$\$dim:A_0\$ = 6$

$\$A[[i]][j]][k]\$ = (i*8 + j*4 + k)$

A[[0][0]][0]	A[[0][0]][1]	A[[0][0]][2]	A[[0][0]][3]
A[[0][1]][0]	A[[0][1]][1]	A[[0][1]][2]	A[[0][1]][3]
A[[1][0]][0]	A[[1][0]][1]	A[[1][0]][2]	A[[1][0]][3]
A[[1][1]][0]	A[[1][1]][1]	A[[1][1]][2]	A[[1][1]][3]
A[[2][0]][0]	A[[2][0]][1]	A[[2][0]][2]	A[[2][0]][3]
A[[2][1]][0]	A[[2][1]][1]	A[[2][1]][2]	A[[2][1]][3]



Subarrays

```
$B [[3] [2]] [4] @DMEM(float)$
```

```
float* y;
```

```
$create(y):B$;
```

```
CONT_TRANSFER(y, $B$, x, $A$)
```

Device Memory

Contiguous data transfer : cudaMemcpy().

```
$C / B[_1][_0]$
```

```
DISC_TRANSFER(y, $C$, x, $A$)
```

```
$D / B[[_0_1][_0_0]][_1]$
```

```
MIXED_TRANSFER(y, $D$, x, $A$)
```

Transposed subarray

Sub-dimensions swapped

```
$E / A[_1][_0_0]$
```

E[0][0]	E[1][0]	E[2][0]	E[3][0]
E[0][1]	E[1][1]	E[2][1]	E[3][1]
E[0][2]	E[1][2]	E[2][2]	E[3][2]



Library of Sub-

Programs

```
/*parray_define ALLTOALL(target, source, type) {
  $IDTYPE[$dim:type_0$]@PAGED(int)$
  $SIDTYPE[$dim:type_0$] []/IDTYPE[dummy] []$
  $TIDTYPE[$dim:type_0$] []/IDTYPE[dummy] []$
  include DIST_GROUP_TRANSFER($TIDTYPE$, target, type, $SIDTYPE$,
    source, type, CONT_TRANSFER) {}
}
parray_end*/
```

Dummy dimension

Collectives also work for
CPU shared mem!

```
/*parray_define SETS_TO_SETS(idtype, target, source, type) {
  if ($dim$ != $dim:idtype$ )
    {printf("Invalid number of threads in collective communication.\n");exit(-1);}
  $IDTYPE[[] [$dim:idtype_1$]] []/idtype[[_0][dummy]][_1]$
  include DIST_GROUP_TRANSFER($IDTYPE$, target, type, $IDTYPE$,
    source, type, CONT_TRANSFER) {}
}
parray_end*/
```

Abstraction achieved by
calling Parray library.

```
/*parray_define SCATTER(root, target, source, type) {
  $TARGET[$dim:type_0$] [] /type[dummy][_1] !const long$
  $SIDT[(($tid$==root?$dim:type_0$:0)]@PAGED(int)$
  $TIDT[1]@PAGED(int)$
  $SIDTYPE[$dim:type_0$] []/SIDT[dummy] []$
  $TIDTYPE[$dim:type_0$] []/TIDT[exp(root)] []$
  include DIST_GROUP_TRANSFER($TIDTYPE$, target, $TARGET$, $SIDTYPE$,
    source, type, CONT_TRANSFER) {}
}
parray_end*/
```

Single Program, Multiple Code-blocks

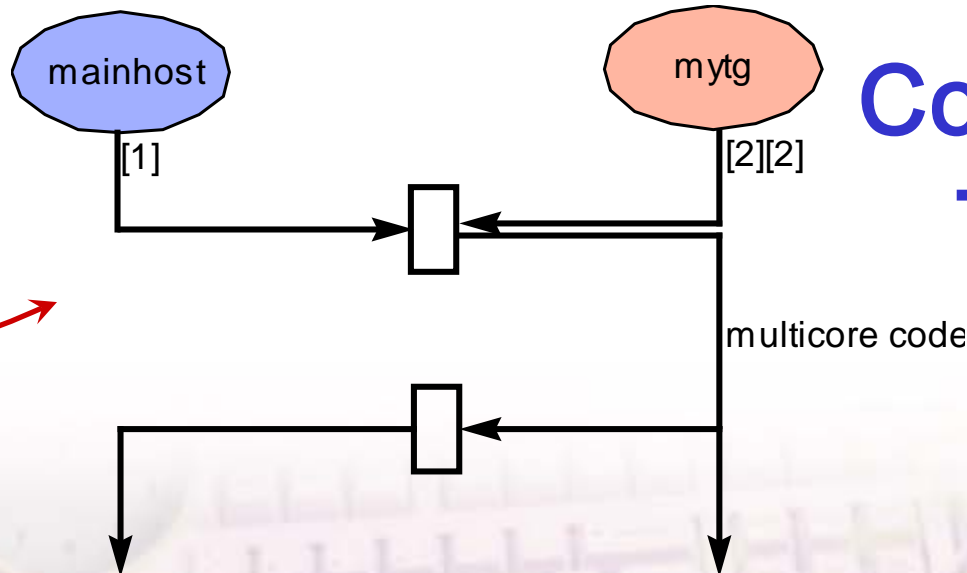
```
mainhost{
  $MYTG[2][2]@PTHREAD_TG$
  pthread_tg* mytg;
  $create(mytg):MYTG$;
  detour(, $tg(mytg):MYTG$) {
    printf("Hello world %d,%d\n", $tid_0$, $tid_1$);
  }
  $destroy(mytg):MYTG$;
}
```

multicore thread group

pthread_create()

Thread row id

pthread_join()

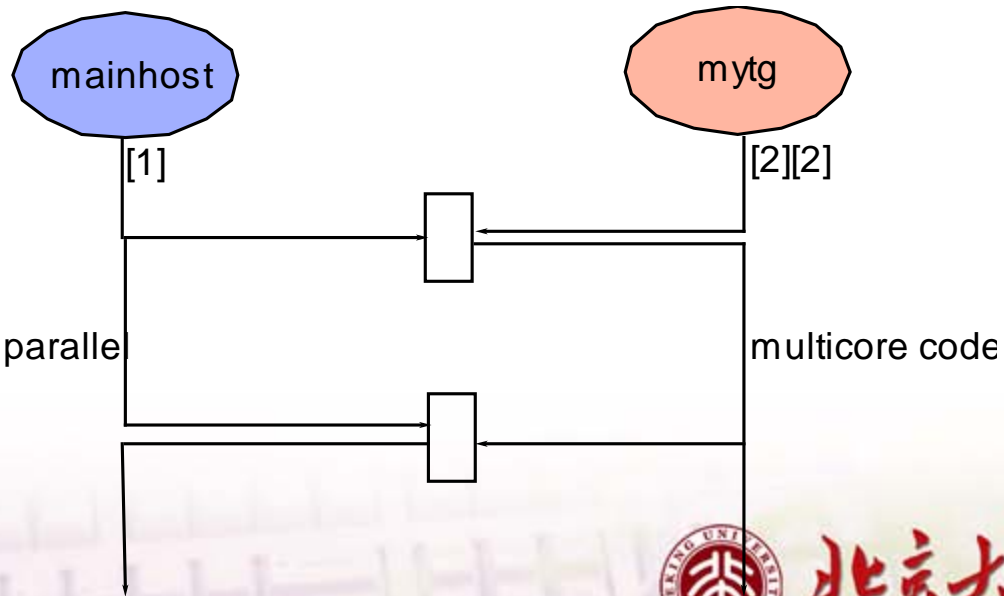


Control Flow Travels!



Two Parallel Detours

```
mainhost{
  $MYTG[2][2]@PTHREAD_TG$
  pthread_tg* mytg;
  $create(mytg):MYTG$;
  parallel{
    detour(, $tg(mytg): MYTG$) {
      printf("Hello world %d,%d\n", $tid_0$, $tid_1$);}
    detour(, _mainhost) {
      printf("mainhost: hello\n");}
  }
  $destroy(mytg):MYTG$;
}
```



```
__global__ void sgemmNN( const float *A, int lda, const float *B, int ldb, float* C, int ldc, int k, float alpha, float beta )
```

```
{
```

```
    A += blockDim.x * 64 + threadIdx.x + threadIdx.y*16;  
    B += threadIdx.x + ( blockDim.y * 16 + threadIdx.y ) * ldb;  
    C += blockDim.x * 64 + threadIdx.x + (threadIdx.y + blockDim.y * ldc ) * 16;
```

} Compute pointers to the data

```
    __shared__ float bs[16][17];  
    float c[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};  
    const float *Blast = B + k;
```

} Declare the on-chip storage

CUDA SGEMM

(CUBLAS 2.0 Volkov SC'08)

```
    do  
    {  
#pragma unroll  
        for( int i = 0; i < 16; i += 4 )  
            bs[threadIdx.x][threadIdx.y+i] = B[i*ldb];  
        B += 16;  
        __syncthreads();
```

} Read next B's block

```
#pragma unroll  
    for( int i = 0; i < 16; i++, A += lda )  
    {  
        c[0] += A[0]*bs[i][0];  c[1] += A[0]*bs[i][1];  c[2] += A[0]*bs[i][2];  c[3] += A[0]*bs[i][3];  
        c[4] += A[0]*bs[i][4];  c[5] += A[0]*bs[i][5];  c[6] += A[0]*bs[i][6];  c[7] += A[0]*bs[i][7];  
        c[8] += A[0]*bs[i][8];  c[9] += A[0]*bs[i][9];  c[10] += A[0]*bs[i][10];c[11] += A[0]*bs[i][11];  
        c[12] += A[0]*bs[i][12];c[13] += A[0]*bs[i][13];c[14] += A[0]*bs[i][14];c[15] += A[0]*bs[i][15];  
    }  
    __syncthreads();
```

} The bottleneck:
Read A's columns
Do Rank-1 updates

```
    } while( B < Blast );  
    for( int i = 0; i < 16; i++, C += ldc )  
        C[0] = alpha*c[i] + beta*C[0];
```

} Store C's block to memory

```
}
```

Example: SGEMM

```
$MYTG[[YBLOCKS][XBLOCKS]][[4][32]] @CUDATG(float* A, float* B, float* C)$
```

```
$ATYPE[[YBLOCKS][[4][4]]] [[MATRIX_WIDTH/32][32]] /type[_0][_1]$
```

```
$AOUTER_TYPE /ATYPE[_0_0][[_0_1_1][_1_1]]$
```

manycore thread group

```
$BTYPE[[MATRIX_WIDTH/32][32]][[XBLOCKS][[128][2]]] /type[_0][_1]$
```

```
$BOUTER_TYPE /BTYPE[_1_0][_1_1_0]$
```

```
$CTYPE[[YBLOCKS][16]][[XBLOCKS][[128][2]]] /type[_0][_1]$
```

```
$CINNER /CTYPE[_0_1][_1_1_1]$
```

```
$DTYPE[32][17] @SMEM(float)$
```

```
$DBANK_TYPE[32][[4][4]] /DTYPE[_0][_1] $
```

Handling bank conflict in shared mem

```
$GTYPE[16][2] @RMEM(float)$ }
```

```
detour(, _kernel($MYTG$, (d_a)(d_b)(d_c))){
```

Launch kernel

```
    $create(D):DTYPE$; $create(G):GTYPE$; int tid1 = $tid_1$;
```

```
    #pragma unroll
```

```
    for (int i=0; i<$dim:GTYPE$; i++) G[i]=0;
```

```
    args.A += $AOUTER_TYPE[$tid_0_0$][tid1]$;
```

```
    args.B += $BOUTER_TYPE[$tid_0_1$][tid1]$;
```

```
    args.C += $CTYPE[[ $tid_0_0$ ][0]][[ $tid_0_1$ ][[tid1][0]]]$;
```

Read

```
    for (int i=0; i<$dim_1_0:ATYPE$; i++, args.A+=$dim_1_1:ATYPE$){
```

```
        include DISC_TRANSFER(D+$DBANK_TYPE[$tid_1_1$][[0][ $tid_1_0$]]$, $DBANK_TYPE_1_0$,  
            args.A, $ATYPE_0_1_0$){}
```

```
        include SYNC(){}
```

```
        #pragma unroll
```

```
        for (int m=0; m<32; m++){
```

```
            float2 b = ((float2 *) (args.B+ $BTYPE_0[i][m]$))[0];
```

```
            int j=0; UNROLL(16, G[$GTYPE[j][0]$] += b.x*D[$DTYPE[m][j]$];
```

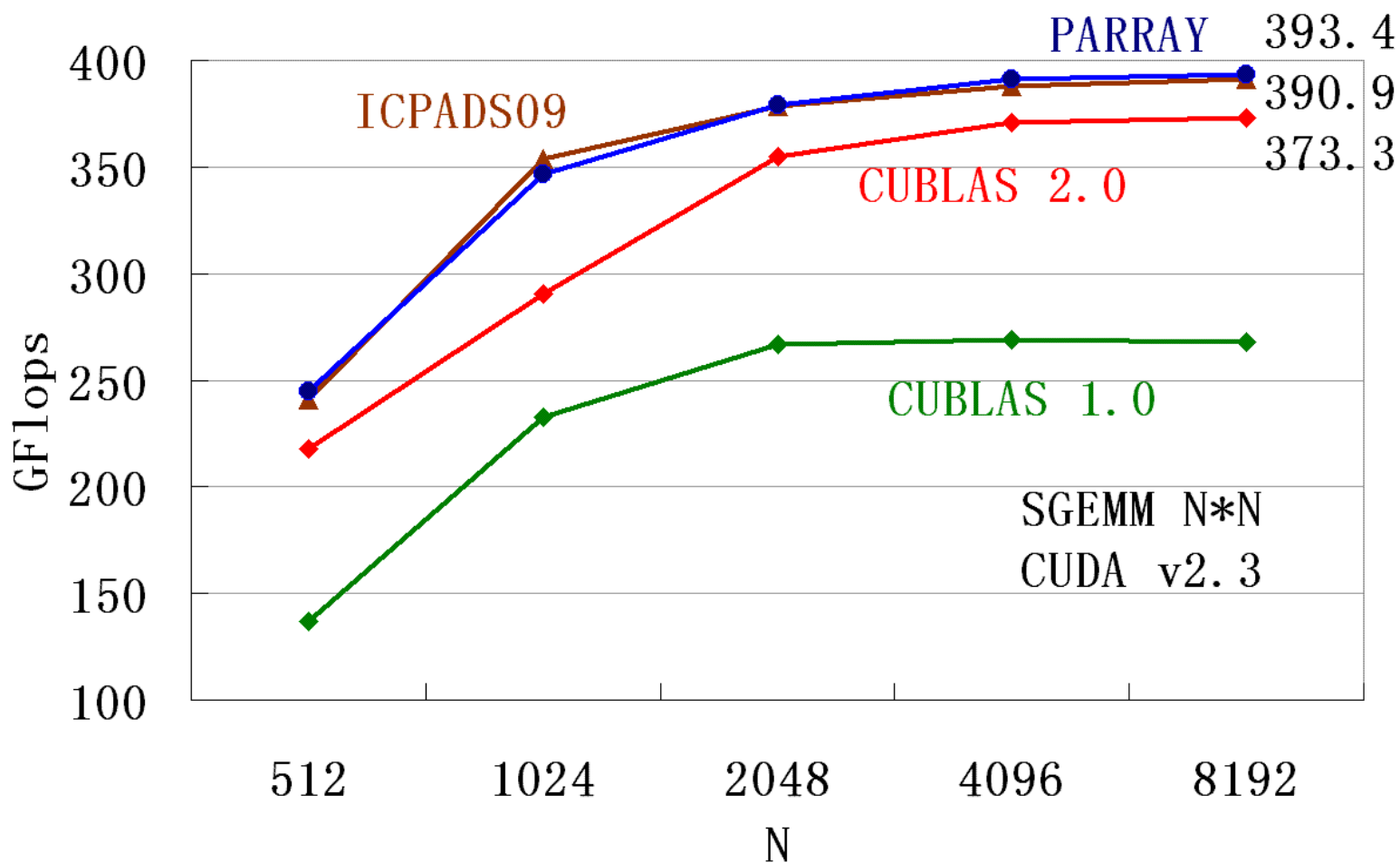
```
                G[$GTYPE[j][1]$] += b.y*D[$DTYPE[m][j]$]; , j++;)
```

Compute

```
            include SYNC(){}
```

```
        include DISC_TRANSFER(args.C, $CINNER$, G, $GTYPE$){} }
```

Write



Tesla C1060



北京大学
PEKING UNIVERSITY

Example: Non-Portable PKUFFT (>500 Lines, 24x vs. FFTW)

```
for (int round=0;round<ROUND;round++){  
  // ----- pinned hmem to dmem -----  
  for (int i = 0; i < 4; i++){ cudaMemcpyAsync(gpumem+i*NUM128MTYPED,  
    pin_mem[gpuid]+round*NUM512MTYPED+i*NUM128MTYPED,  
    NUM128M, cudaMemcpyHostToDevice,stream[i] ); cufftSetStream(plan,stream[i]);  
    cufftExecC2C( plan, gpumem+i*NUM128MTYPED, gpumem+i*NUM128MTYPED,  
    CUFFT_FORWARD );}  
    cudaThreadSynchronize();  
  // ----- dmem to paged hmem -----  
  for (int j=0;j<4;j+=2){  
    if (round>0) sem_wait(&semCopied[gpuid]);  
    for (int i=0;i<2;i++){  
      .....  
      cudaMemcpy(gpumem+(i+j)*NUM128MTYPED+NUM1M*(myrank+1),NUM8M*(15-myrank),  
        cudaMemcpyDeviceToHost); }  
    sem_post(&semGPUsyn[gpuid]); sem_wait(&semGPUsyn[1-gpuid]);  
    if (!(round==0 && j==0)) sem_post(&semToCopy[gpuid]);  
    // ----- all to all communication -----  
    for (int i=0; i<2 ; i++){ ib[gpuid].rem_dest = all_rem_dests[gpuid][i+j];  
      ib[gpuid].data_transfer(myrank,i+j,all_my_keys[gpuid][i+j]); }  
    // ----- all threads sync -----  
    sem_post(&semGPUsyn[gpuid]); sem_wait(&semGPUsyn[1-gpuid]);  
    if (gpuid==0) MPI_Barrier(MPI_COMM_WORLD);  
    sem_post(&semGPUsyn[gpuid]); sem_wait(&semGPUsyn[1-gpuid]); }  
}
```

CUDA calls

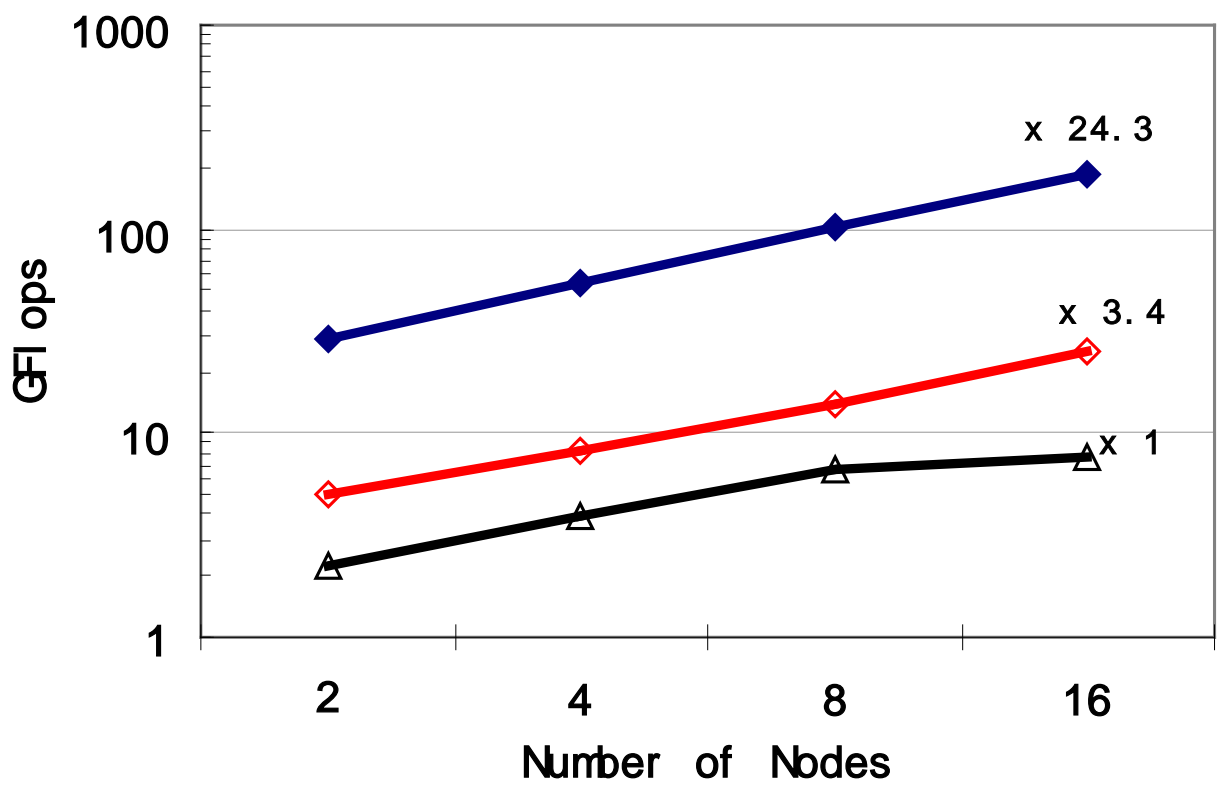
Pthread calls

Infiniband/verbs calls

MPI calls



北京大學
PEKING UNIVERSITY

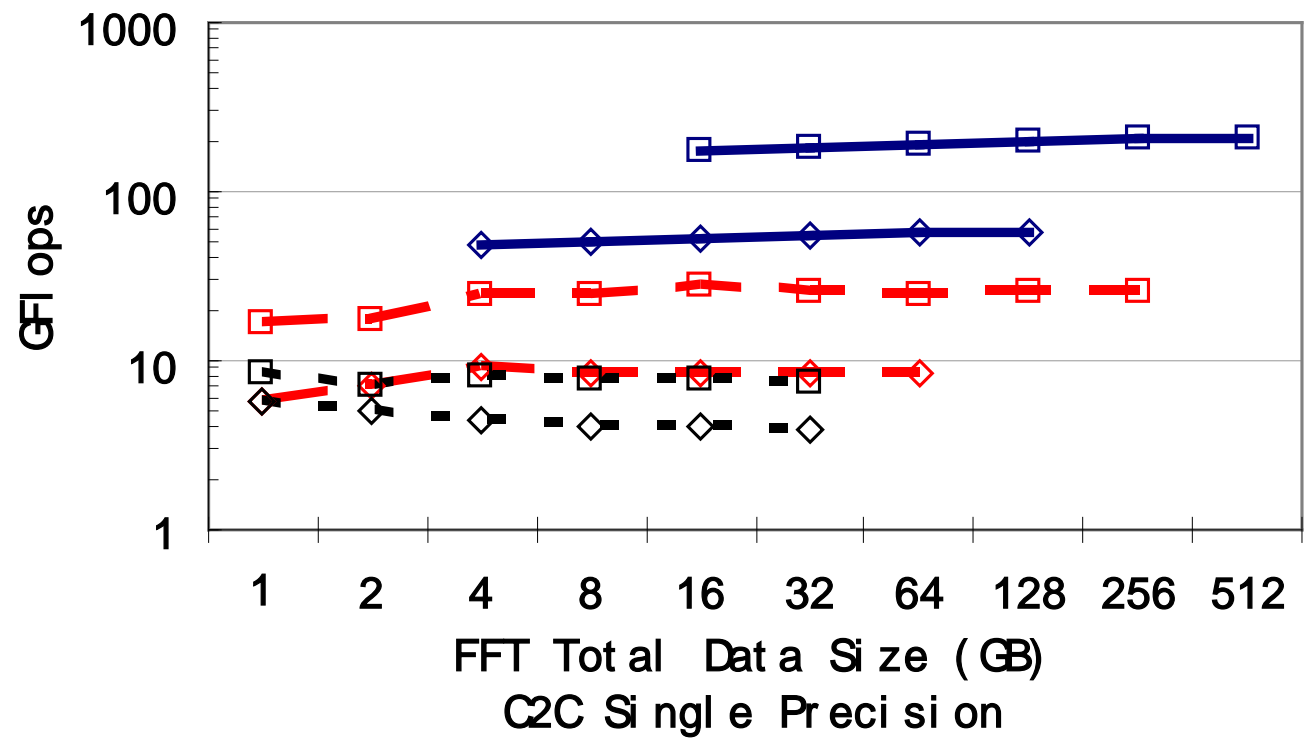
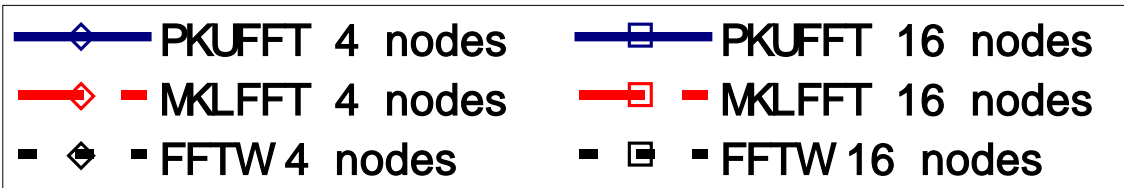


C2C Single Precision Data 32GB

ICS'10



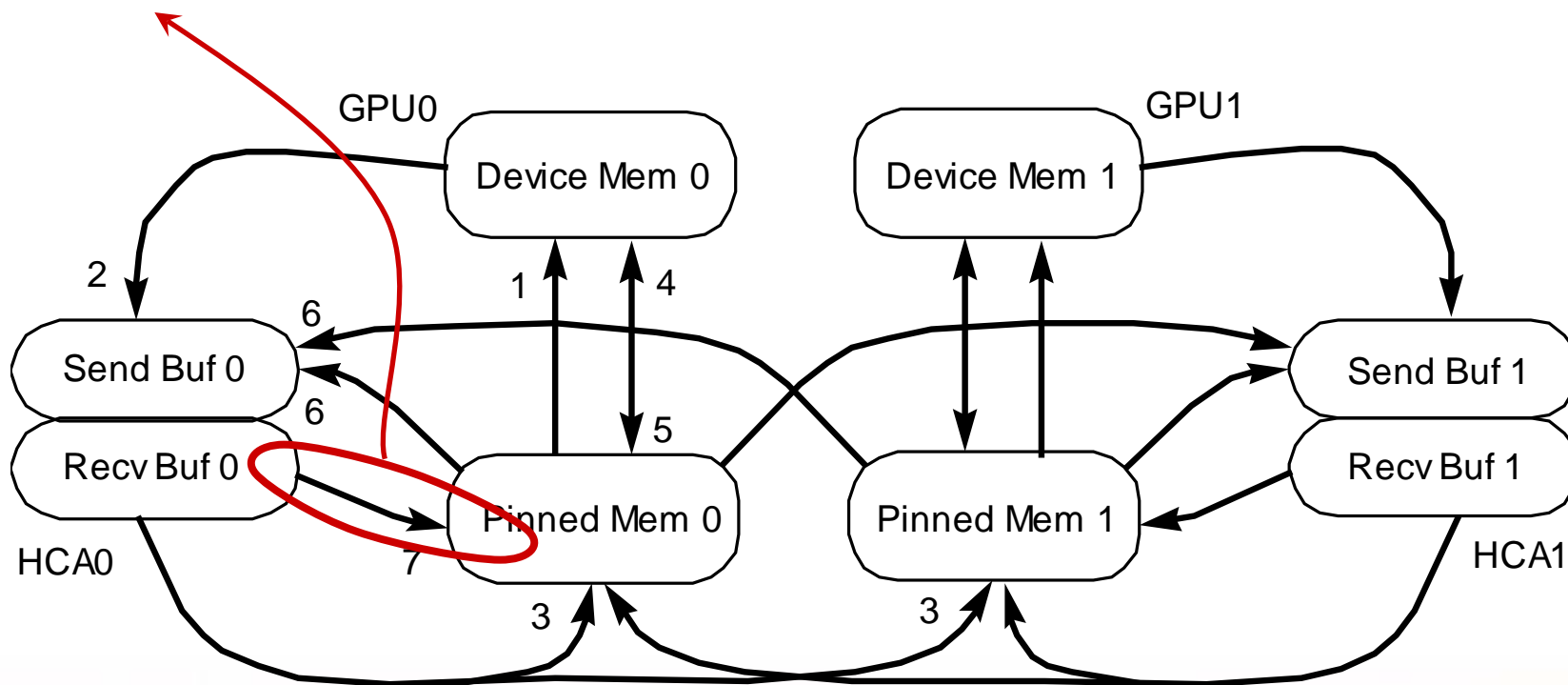
北京大学
PEKING UNIVERSITY



576GB Total memory max processing
512GB data for 4096 3D C2C FFT



```
for (int i=0; i<2; i++) for (int j=0; j<32; j++)  
for (int k=0; k<2; k++) for (int l=0; l<16; l++)  
memcpy(pin_mem[i]+round*NUM512MTYPED+ (j*128+l*8+ibid*4+half*2+k)*(4*4096),  
paged_mem[ibid]+half*NUM512MTYPED+((2*k+1)*1024+l*64+i*32+j)*(4*4096),NUM128K);
```



Example: Parray FFT (100 lines)

```
/*parray_define TRANSPOSE(adr, type)
  $MYCUDA_TG[[X/TBSIZE][X/TBSIZE]][[TBSIZE][TBSIZE]] @ CUDA_TG(float2* work)$
.....
parray_end*/

/*parray_define MAIN()
public_consts{
  $MYMPITG[NODE_NUM] @MPI_TG$
  $MYPTHDTG[GPU_NUM] @PTHREAD_TG$
  $BUFFER_T[[W][Z1][Z3]][[Y2][Y3][X]] @MPIMEM(float2)$
  $IN_BUF_T /BUFFER_T [[_1_1][_0_0][_0_1_0][_0_1_1]][_1_2]$
  $NEW_SLICE_T /NEW_HOST_T[[_0_1][_1_0]][_1_1]$
.....}
public_vars{ mpi_tg* mympitg; }
localhost{
  pthread_tg* mypthdtg;
  float2* host[GPU_NUM]; float2* slice[GPU_NUM]; float2* dev[GPU_NUM];
  float2* out_buf; float2* in_buf;}
mainhost{
  $create(Public.mympitg):MYMPITG$;
  $detour(, $tg(Public.mympitg):MYMPITG$){
    $create(mypthdtg):MYPTHDTG$; $create(out_buf):BUFFER_T$; $create(in_buf):BUFFER_T$;
    $detour(_every(), $tg(mypthdtg):MYPTHDTG$){
      $create(host[$tid$]):HOST_T$; $create(dev[$tid$]):DEV_T$;
.....
    for(int round=0; round<Z2; round++) {
.....
      include MIXED_TRANSFER(slice[$tid$], $NEW_SLICE_T$, in_buf+$BUFFER_T_1_0[$tid$]$, $IN_BUF_T$){}
.....
    }
    $destroy(mypthdtg):MYPTHDTG$; }}
parray_end*/
```

Manycore thread group

MPI thread group

Pthread thread group

From device memory to
IB buffer

memcpy





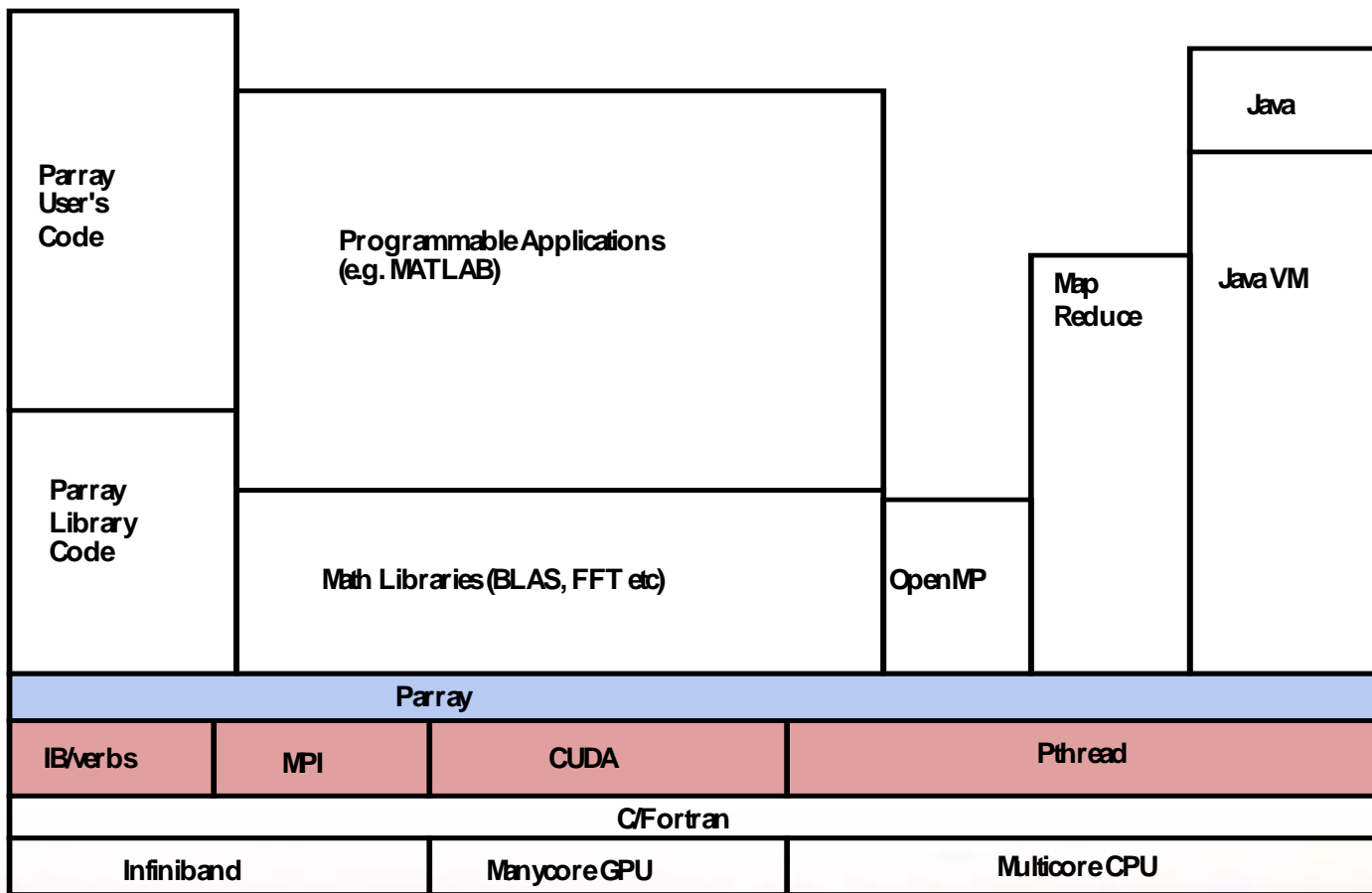
Conclusions and Future Work

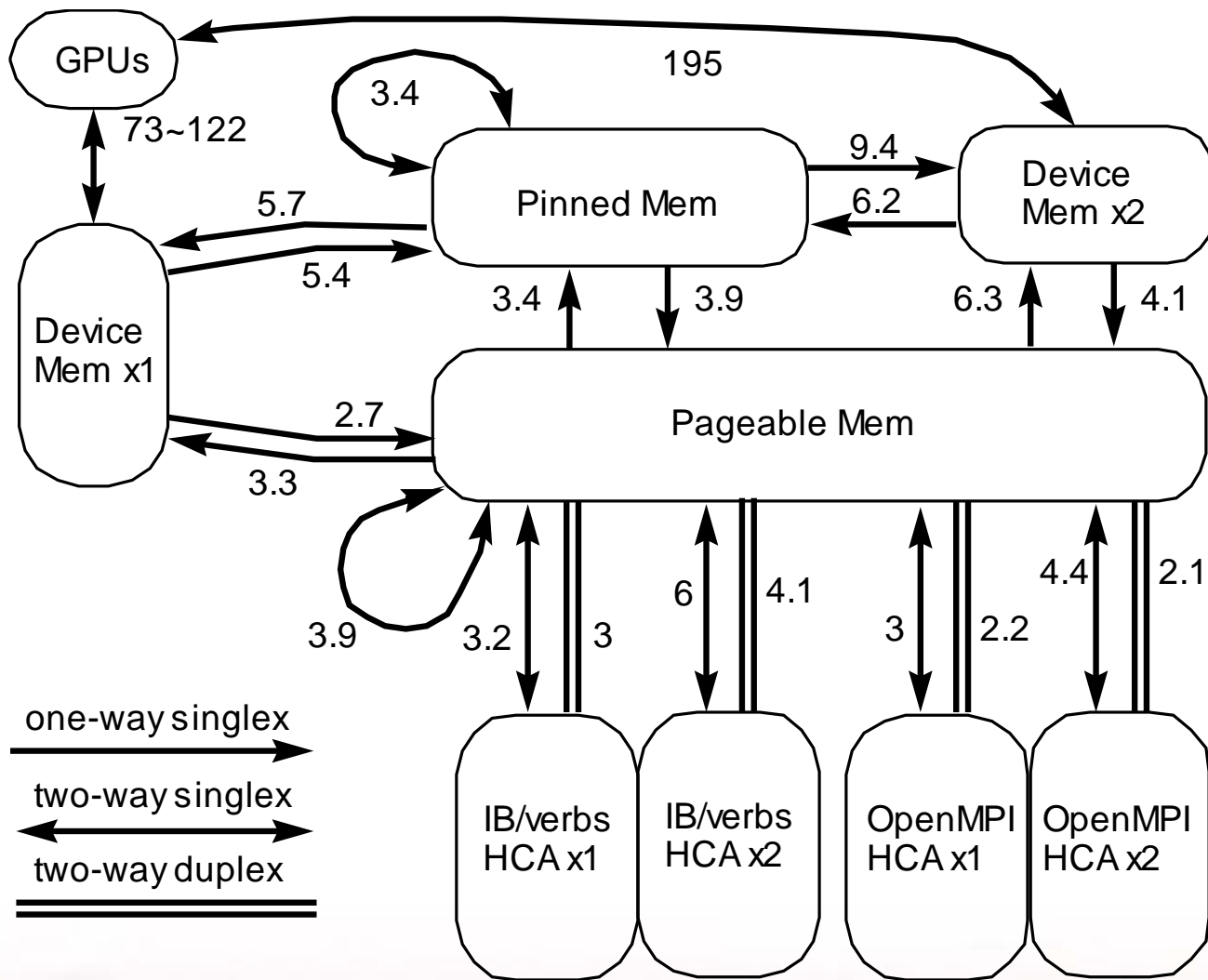
- ◆ Generate all possible communications with little overhead.
- ◆ Compiler community and programming-language community should work together (**need faith in parallel language design**).
- ◆ Direct code generation to IB/verbs in future (**skipping MPI**).



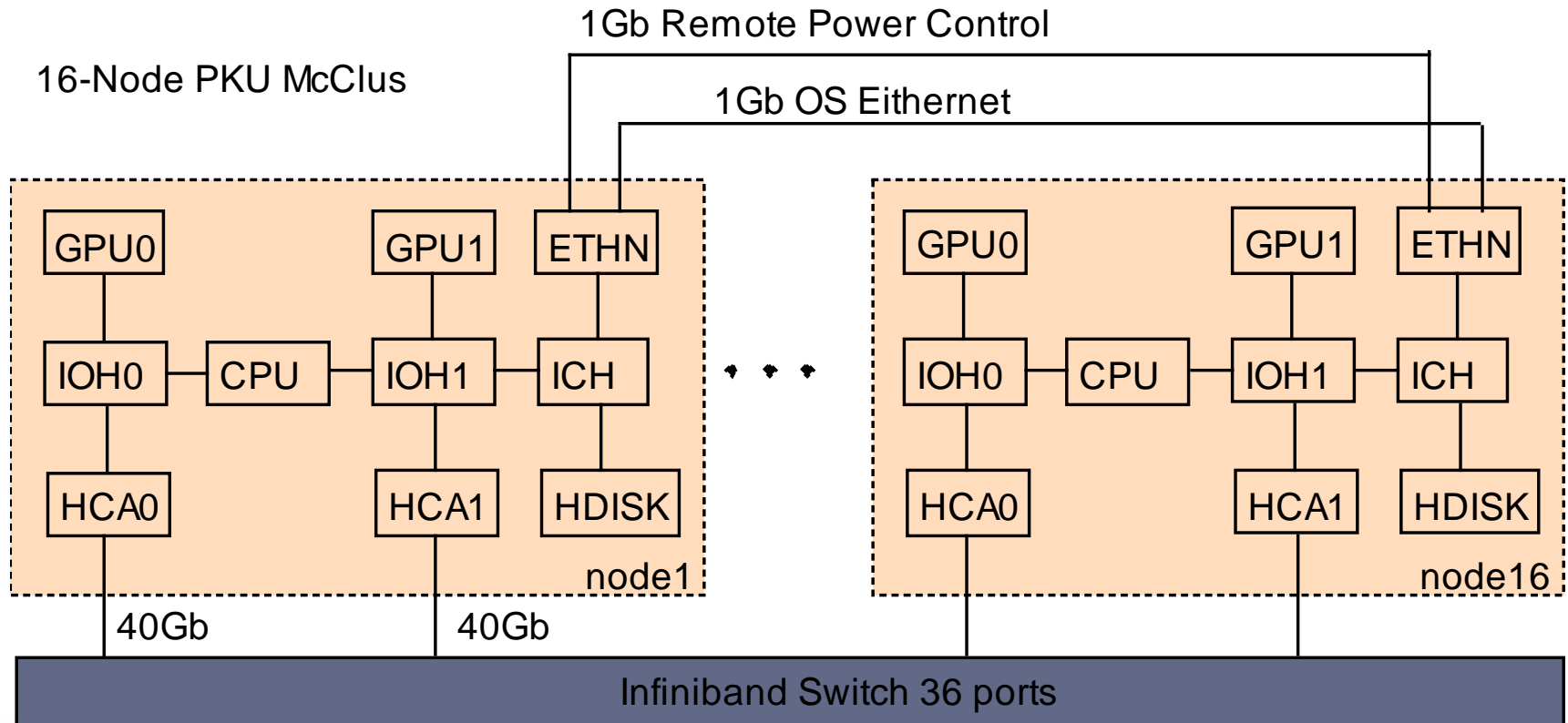


Less exposure of performance-critical features of architecture
Fewer choices of communication and synchronisation
Lower performance in general





PKU McClus 集群结构





The Trend of HPC

◆ What we want:

- high performance
- cheap hardware
- small energy bill.

◆ What we will get:

- deep memory hierarchy
- different processor cores (GPU and CPU)
- millions of them.

◆ This trend is likely to continue!

