Acceleration of Streamed Tensor Contraction Expressions on GPGPU-based Clusters

Wenjing Ma[1], Sriram Krishnamoorthy[2], **Oreste Villa** [2], Karol Kowalski[2]

[1] Ohio State University, Columbus, Ohio.

[2] Pacific Northwest National Laboratory, Richland, WA.

mawe@cse.ohio-state.edu , sriram@pnl.gov, oreste.villa@pnl.gov karol.kowalski@pnl.gov

PNNL – LDRD Exascale Initiative

Outline

- Motivation and Introduction
  - NWChem and CCSD
  - GPU clusters
- Tensor Contraction (TC)
- GPU implementation of TC
  - Basic implementation
  - Index combining
  - Flattening
  - Streaming
- Evaluation and experimental results
- Discussion on future systems
- Conclusion and future work

NWChem and CCSD(T)

- NWChem
  - Computational Chemist application developed at Pacific Northwest National Laboratory (PNNL)
  - One of the methods involves the solution of the time independent Schrodinger equation

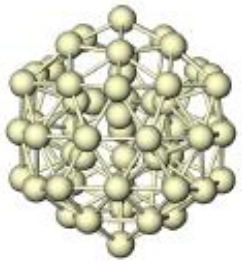
$$\hat{H}\ \mathrm{e}^{\hat{T}}|\Psi_0\rangle \quad = \quad E\ \mathrm{e}^{\hat{T}}|\Psi_0\rangle$$

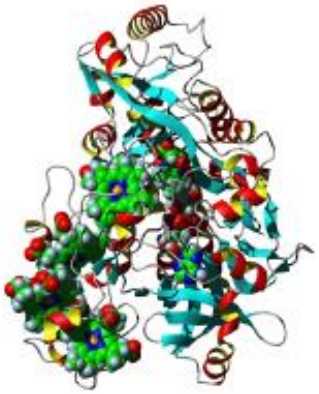
$$\hat{T} \quad = \quad \hat{T}_1 + \hat{T}_2 + \hat{T}_3 + \cdots + \hat{T}_N = \sum_{p}^{N} \hat{T}_p$$

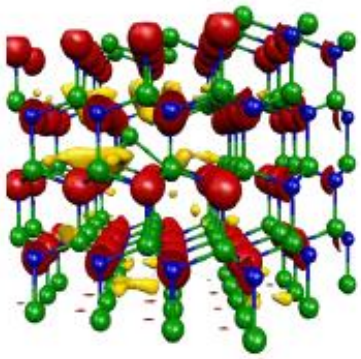
  - The method is called CCSD(T) and consist in calculating the Cluster operator T up to the third contribution
    - Coupled Cluster Single Double (with Triple correction)
  - $T_1 = 0(n^{1 * 2 + 1})$ - $T_2 = 0(n^{2 * 2 + 1})$ - $T_3 = 0(n^{3 * 2 + 1})$

One of the "Few" PetaFlops applications

- CCSD(T) theory is used to understand fundamental optical processes in solar cells, photosynthesis, and other optically active materials.
- ORNL's Jaguar achieve 1.31 PFlops (over 50% of peak) on 225,000 processors during CCSD calculation
  - ***ORNL –PNNL Collaboration Lead Edo Apra*** (Gordon Bell Finalist at SC 2009)







Pacific Northwest
NATIONAL LABORATORY

GPU Accelerated Clusters (from top500.org)

Rank	Site	Computer
1	Oak Ridge National Laboratory United States	Jaguar - Cray XT5-HE Opteron Six Core 2.6 GHz Cray Inc.
2	National Supercomputing Centre in Shenzhen (NSCS) China	Nebulae - Dawning TC3600 Blade, Intel X5650, NVidia Tesla C2050 GPU Dawning
3	DOE/NNSA/LANL United States	Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband IBM
4	National Institute for Computational Sciences/University of Tennessee United States	Kraken XT5 - Cray XT5-HE Opteron Six Core 2.6 GHz Cray Inc.
5	Forschungszentrum Juelich (FZJ) Germany	JUGENE - Blue Gene/P Solution IBM
6	NASA/Ames Research Center/NAS United States	Pleiades - SGI Altix ICE 8200EX/8400EX, Xeon HT QC 3.0/Xeon Westmere 2.93 Ghz, Infiniband SGI
7	National SuperComputer Center in Tianjin/NUDT China	Tianhe-1 - NUDT TH-1 Cluster, Xeon E5540/E5450, ATI Radeon HD 4870 2, Infiniband NUDT

ORNL Jaguar in 2012 ~20 PFlops with Nvidia “Fermi” GPUs

Pacific Northwest
NATIONAL LABORATORY

Tensor Contraction (I)

```
R[a,b,c,i,j,k] -= T[l,k,c,b]*V[l,a,i,j]
```

- Tensor contractions are generalized multidimensional matrix multiplication operations that widely occur in quantum chemistry.
- The tensor contraction involves computing tensor R from tensors T and V.
- The index l is common to both input tensors, corresponding to the common dimension in matrix-matrix multiplication, and is referred to as the contracted index.
- CCSD(T) consists of 18 such contraction expressions N^7
- CCSD has 100 types but they are N^5 (types refers to different index permutations)

Tensor Contraction (II)

- Typical computation is a set (several millions) of tiles as follows: (task approach to compute tiles)

```
for ( p4 = 0; p4 < p4d; p4++)
  for ( p5 = 0; p5 < p5d; p5++)
    for ( p6 = 0; p6 < p6d; p6++)
      for ( h1 = 0; h1 < h1d; h1++)
        for ( h2 = 0; h2 < h2d; h2++)
          for ( h3 = 0; h3 < h3d; h3++)
            for ( p7 = 0; p7 < p7d; p7++) {
               C[h3][h2][h1][p6][p5][p4]  -=
                  A[p7][p4][h1][h2] * B[p7][h3][p6][p5];
             }
```

Accumulation

Each dimension is <= 20 → ~ 2GB of temporal tensor per MPI process.

Challenges

- Large tensors + memory requirements of the application, results in each dimension being relatively small.
  - interferes with achieving good locality in SM
  - incurs high index computation overhead (modulo operations)
  - poor device thread block utilization on GPUs
- In general, tensor contraction is harder to efficiently compute than square matrix-matrix multiplication often employed in benchmark studies
- Problem sizes are not known until runtime
- CCSD(T) has 18 contraction types

CUDA Baseline Implementation

```
R[a,b,c,i,j,k] -= T[l,k,c,b]*V[l,a,i,j]
```

a b c l j k l sizes

1. Memory management (due to the tiles)

```
double *devR=allocGPUmem(size_of_R); //similarly devT,devV
```

2. Kernel Arguments
   - common strides offsets pre-computed by the host
3. Encoding Thread-Block Specific Arguments
   - Using CUDA block dimensions to map index dimensions
4. Computation in a Thread-Block
   - One thread computes on element in the output array.
   - The threads in each thread block co-operate in moving data between the GPU memory and shared memory

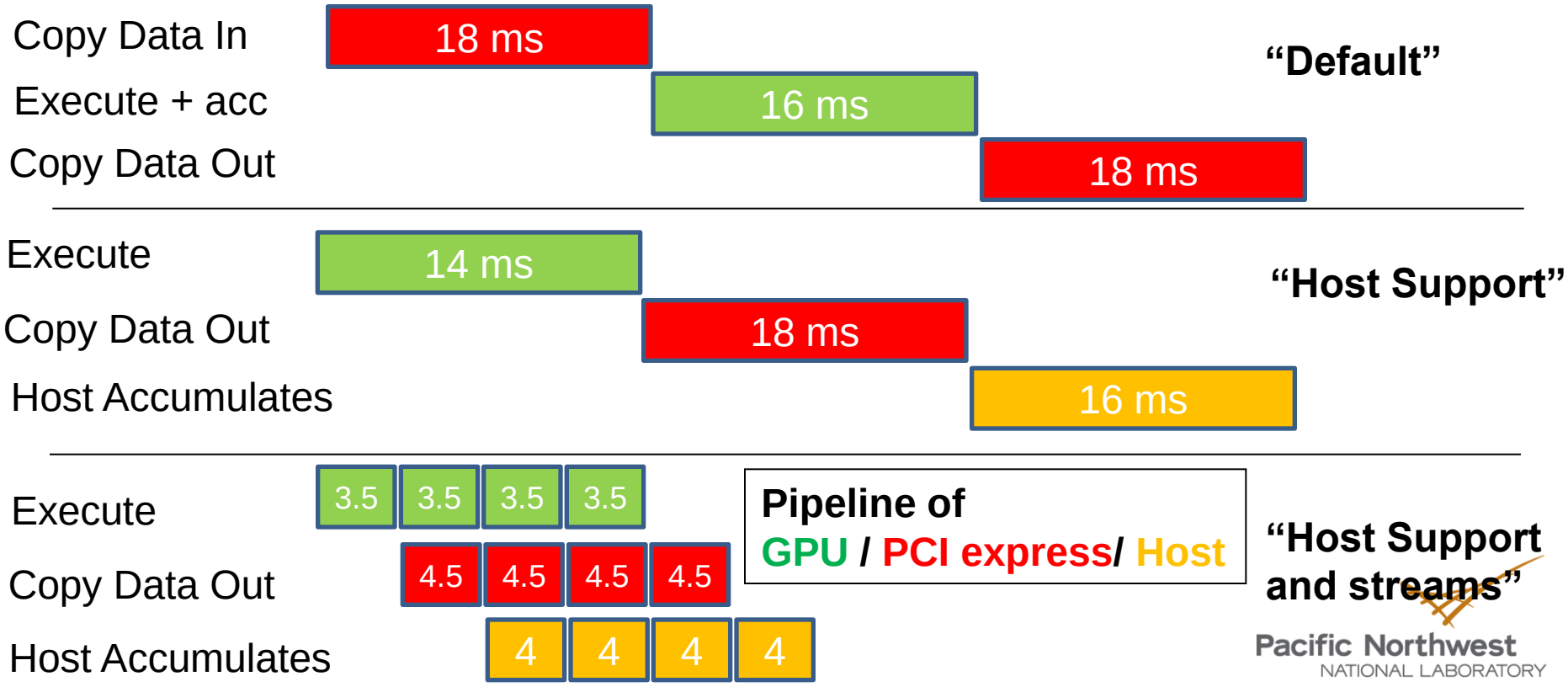
- Default thread Block size 16x16 = 256 Threads

CUDA optimizations

- Index Combining (General optimization)
  - When a pair of indices occur in the same order in every occurrence, they can be replaced by one index whose size is their product → reduction of index calculations
- Dimension Flattening (increase utilization in specific tile sizes)
  - Baseline approach works well only when index dimensions are multiple of the thread block configuration. We flatten the loops and we recreate them of the “right” size, with some index operations → increase of index operations but better utilization
- Pipelining on the outer dimensions (Streaming) with Host Support for accumulation (more details in next slide)
  - Most effective optimization to hide PCI express transfers

Streaming to GPU with CPU accumulation support

- We can avoid the $O(N^6)$ copy IN and just have the copy OUT and then accumulate on the host
- We can create "streams" of kernels using one loop dimension and asynchronously copy out partial data



Experimental System

- 60 nodes cluster named “Barracuda” [1] hosted at PNNL/EMSL
- Each node of “Barracuda” consists of two quad-core Intel Xeon X5560 CPUs with 8MB L2 cache running at 2.80GHz.
- A Tesla S1070 box, which has 4 Tesla 10 GPUs, is shared between two compute nodes, resulting in two GPUs being available per SMP node (tot 128 GPUs)
- Each SMP node is interconnected in the cluster with an Infiniband QDR card.

1. Only interested in double precision (application requirements)
2. We have implemented a Java compiler that starting from the tensor specification generates the CUDA code

[1] *Barracuda was purchased with funds received under the American Recovery and Reinvestment Act*

Single Node Performance (single tile)

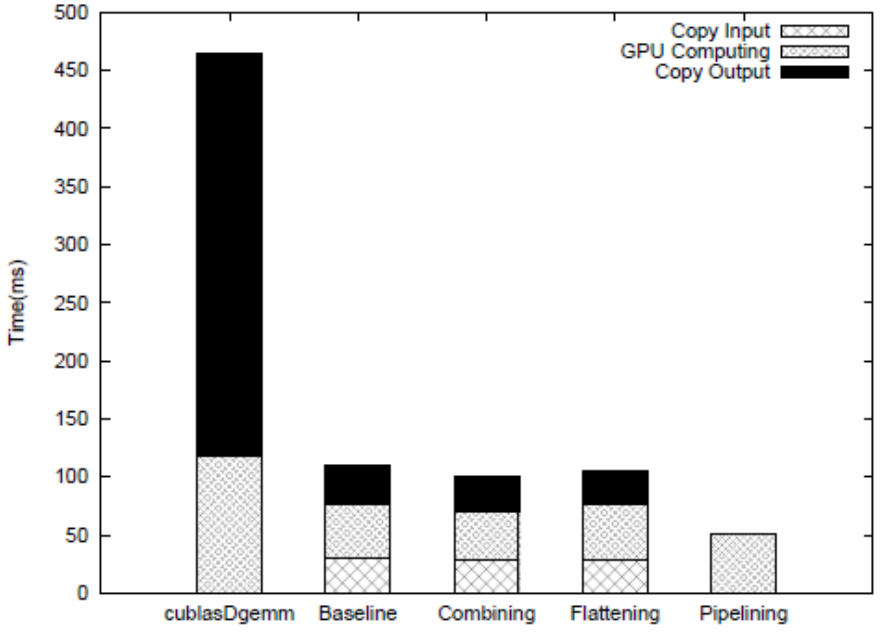


Fig. 1. Sequential benchmark with problem sizes that matches the GPU thread block configuration `(ad=bd=cd=id=jd=kd=ld=16)`

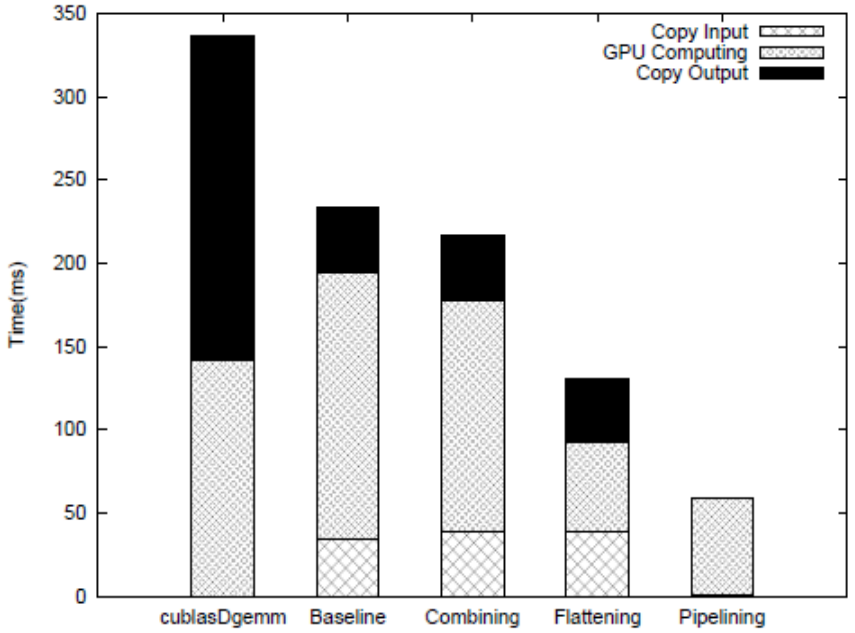


Fig. 2. Sequential benchmark with problem mismatched with GPU thread block configuration `(ad=bd=cd=17, id=jd=kd=16)`

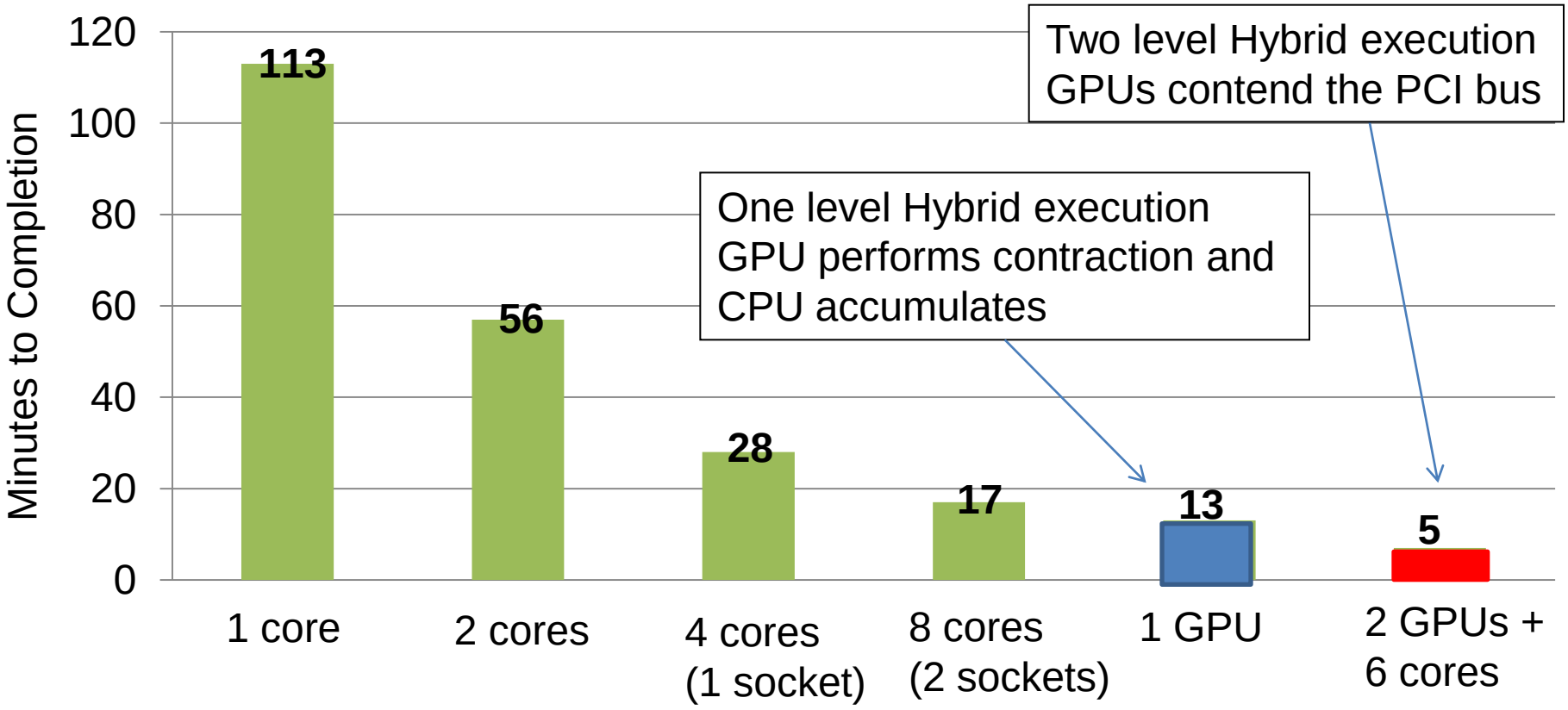
6 dimensions of size 16

3 dimensions of size 16
3 dimensions of size 17

cublasDgemm: (first approach used in the past) In NWChem, each tensor contraction is translated into index permutation and dgemm operations.

Pacific Northwest
NATIONAL LABORATORY

Results for 60 nodes GPU vs CPU (millions of tiles)

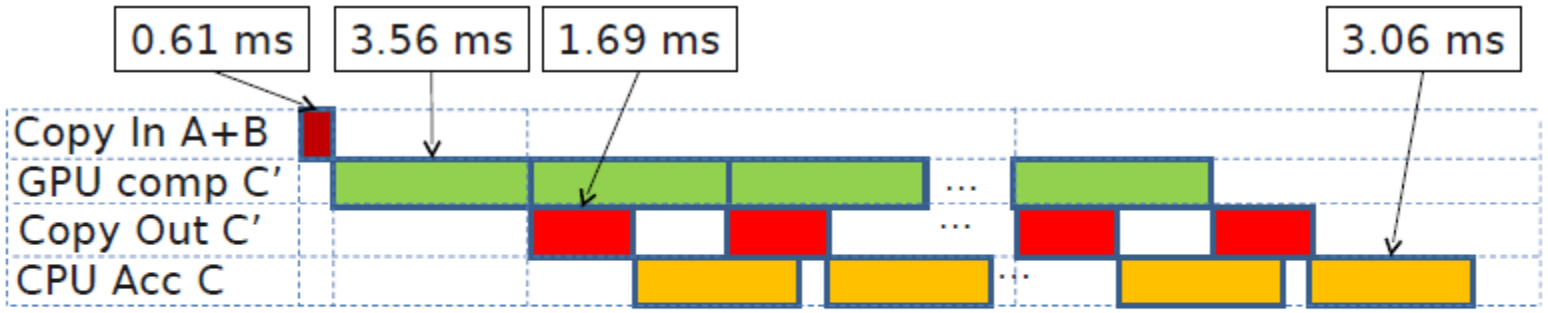


Double precision calculations for green fluorescent protein (GFP) with 284 and 476 basis functions. In all calculations core electrons were not correlated.

Single tile calculation on Tesla C1070

Problem sizes = 17,17,17,16,16,16,16 → Size C = ~160 MB and Num streams =16

A)



Tot time = Copy In A&B + Num streams * GPU comp C' + Copy out C' + CPU comp ACC =
= 0.61 + 16 * 3.56 + 1.69 + 3.06 = 62.31 ms
(Experimental time 64.02 ms)

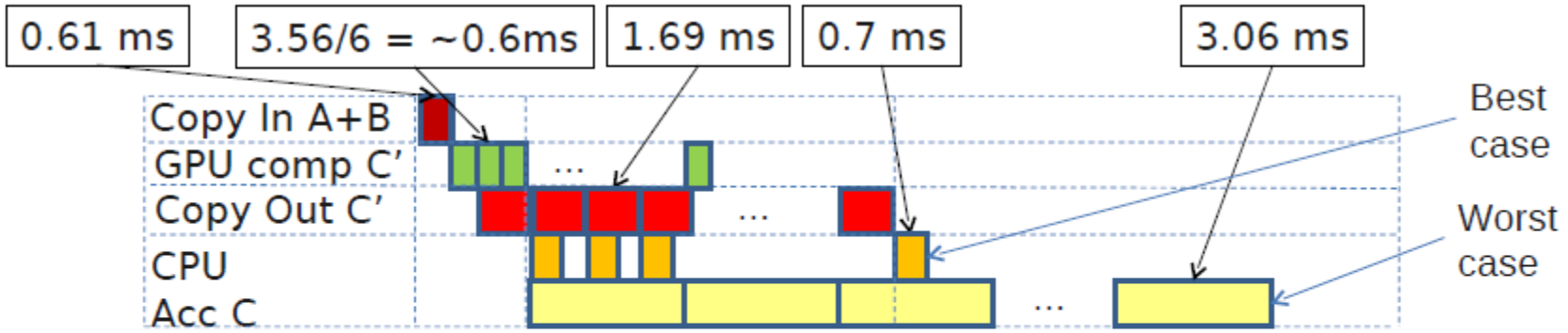
LIMITED BY COMPUTATION

Single tile calculation with “Fermi” with host support (Estimated)

Problem sizes = 17,17,17,16,16,16,16 → Size C = ~160 MB and Num streams =16

Future GPUs Fermi, PCI Exp 2.0 (estimate for streamed version with host support)

B)



Tot time (Worst) = Copy In A&B + GPU comp C' + Copy out C' + Num str. * CPU comp ACC =
= 0.61 + 0.6 + 1.69 + 16 * 3.06 = ~52 ms
(limited by CPU accumulation)

Tot time (Best) = Copy In A&B + Num streams * Copy out C' + CPU comp ACC =
= 0.61 + 16 * 1.69 + 3.06 = ~30 ms
(limited by PCI express transfer)

BEST AND WORST CASE OF CPU ACCUMULATION
Best case only 2X

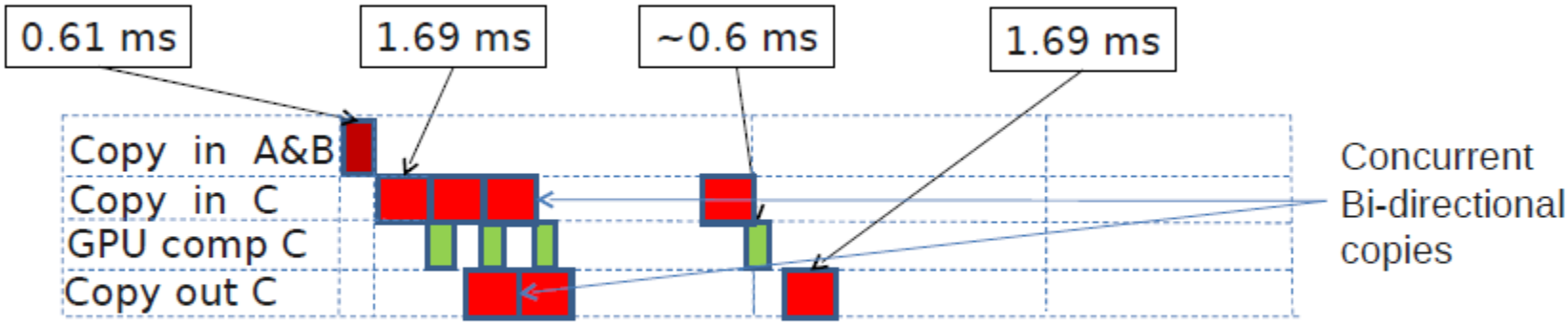
Pacific Northwest NATIONAL LABORATORY

TCE single tile calculation with “Fermi” without host support (Estimated)

Problem sizes = 17,17,17,16,16,16,16 → Size C = ~160 MB and Num streams =16

Future GPUs Fermi and PCI Exp 2.0 (estimate for stream version no host support)

C)



Tot time = Copy In A&B + Num stream * Copy in C + GPU comp C + Copy out C =
= 0.6 + 16 * 1.69 + 0.6 + 1.69 = ~30 ms

If we use this algorithm we are expecting only 2X with “Fermi” respect to previous generation → Confirmed

Pacific Northwest
NATIONAL LABORATORY

Conclusion

- Efficient mapping of tensor contractions to architectural features
- Scheduling through an approach we refer to as dimension flattening.
- Develop efficient pipelined implementations
- Hybrid CPU-GPU execution for tensor contractions
- Evaluation using a full application on a medium-sized GPU cluster
- Evaluation of the proposed approach on possible future GPU-cluster configurations.

Future Work

- Complete the porting of the full CCSD application
  - Copy back free algorithm (high level modification)
  - Iterative part (CCSD) -> up 20% total time
  - Atom to molecule conversion -> up to 30% tot time
  - Execute the application on larger clusters

Thank you!
Questions?

Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by Battelle *Since 1965*