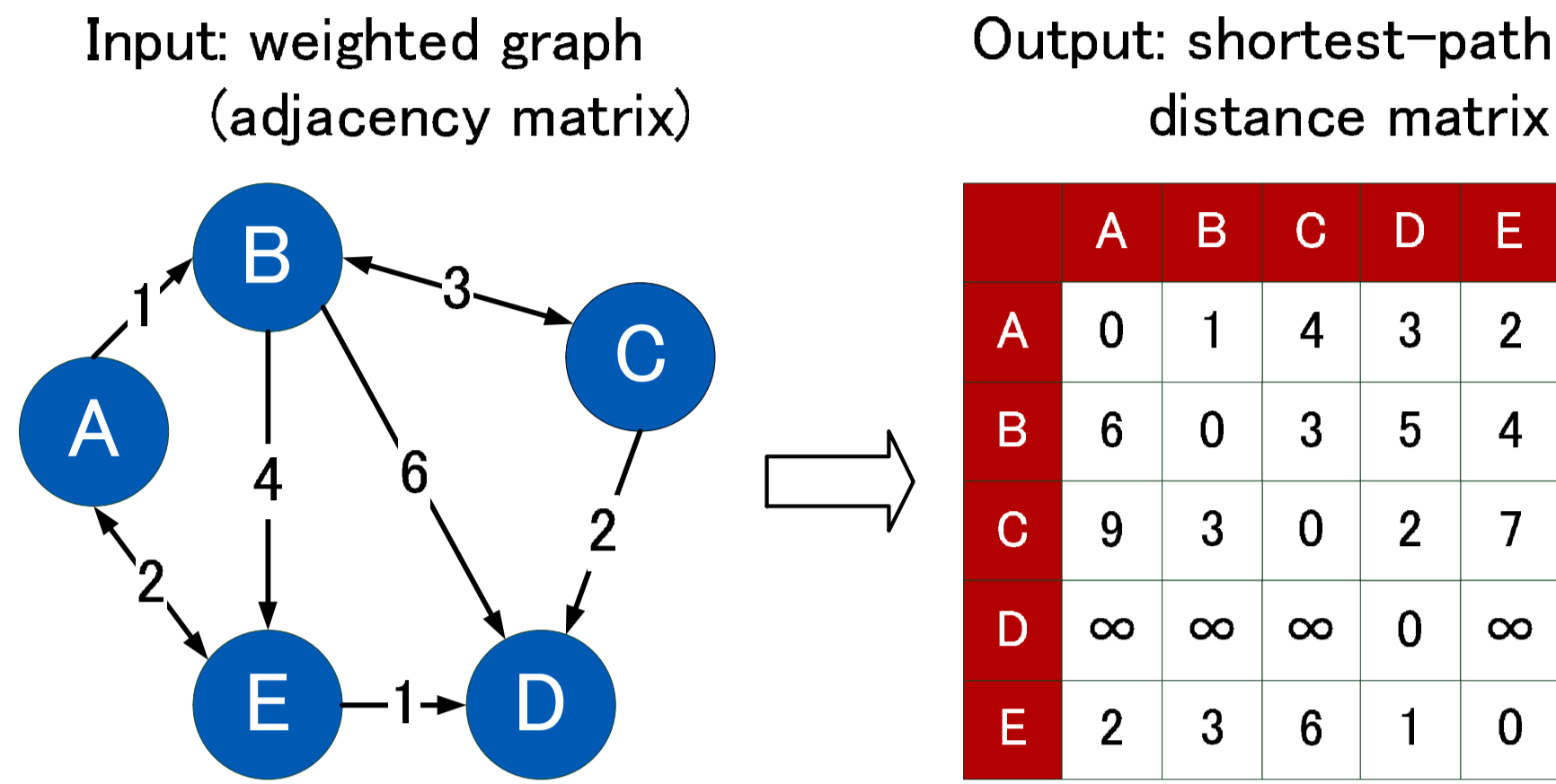


Contribution

- Blocked algorithm for the all-pairs shortest (APSP) paths problem for a hybrid CPU-GPU system.
- Applicable to solve the APSP problem even when the required memory size is larger than GPU's memory capacity.
- Fastest among existing APSP solutions on large dense graphs.
- Discussion on required memory bandwidth for the blocked algorithm.

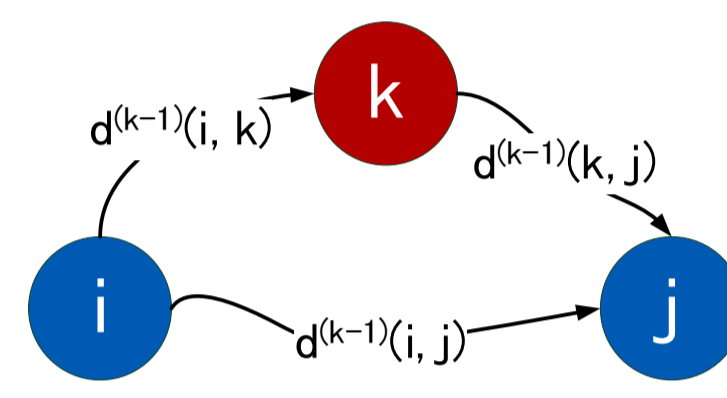
All-Pairs Shortest Paths (APSP) Problem

The all-pairs shortest paths problem is to compute the shortest path length between all-pairs of vertices in a weighted graph. The input can be represented with an adjacency matrix and the output is the shortest-path distance matrix. The APSP problem is one of fundamental graph problems. We can find applications of the APSP problem such as in bioinformatics, social networking, and traffic routing.



Floyd-Warshall Algorithm

- Solution of APSP problem.
- $O(n^3)$ operations on $O(n^2)$ data, where n is the number of vertices.
- Three-nested-loop algorithm structure is similar to the standard matrix multiplication, but has stricter data dependencies.



Algorithm 1 FLOYD-WARSHALL(n, W)

```

 $D^{(0)} \leftarrow W;$ 
for  $k \leftarrow 1$  to  $n$  do
  for all  $1 \leq i \leq n$  do
    for all  $1 \leq j \leq n$  do
       $d^{(k)}(i, j) \leftarrow \min(d^{(k-1)}(i, j), d^{(k-1)}(i, k) + d^{(k-1)}(k, j))$ 
    end for
  end for
end for
return  $D^{(n)}$ ;

```

Blocked APSP Algorithm

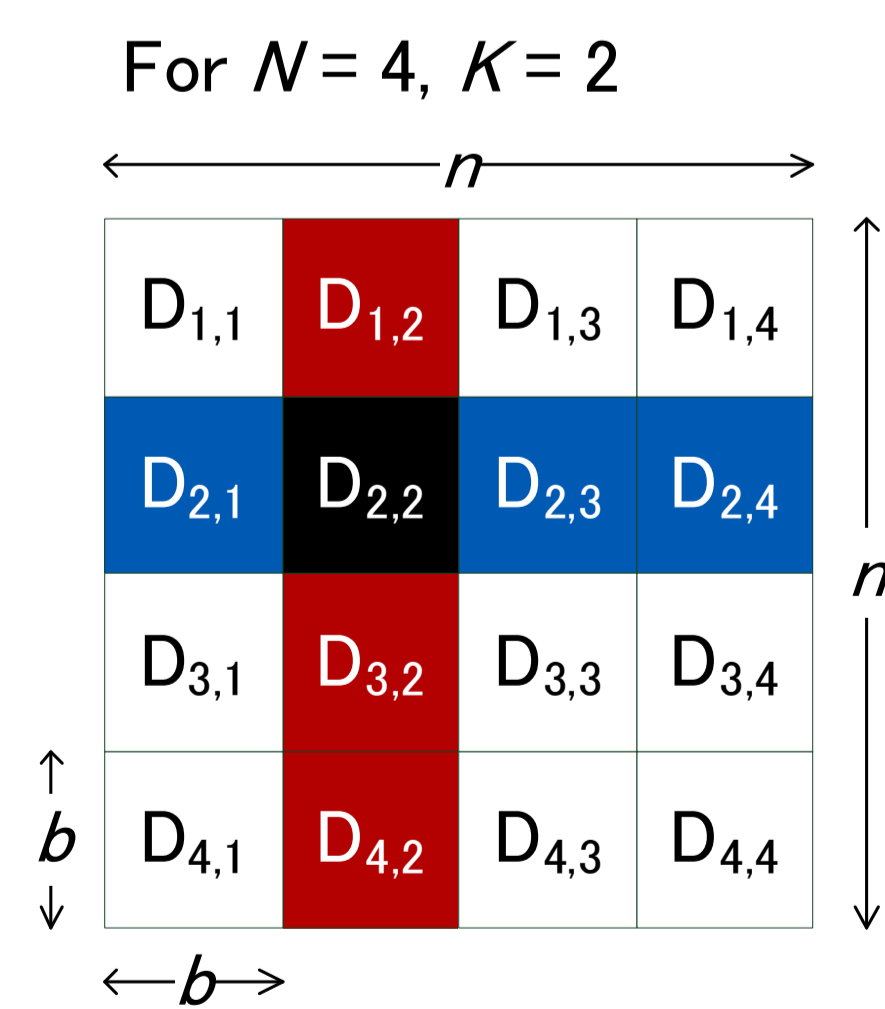
- Partitions an initial $n \times n$ matrix into $(n/b) \times (n/b)$ blocks of $b \times b$ submatrices, where b is a blocking factor.
- Designed for an efficient solution for hybrid systems.
- Most computations are in matrix "multiply-add" (MMA) operations in a min-plus algebra.

Algorithm 2 BLOCKED APSP(n, b, W)

```

Let  $\bar{O}$  be a "zero" matrix whose elements are all  $\infty$ ;
 $N \leftarrow n/b$ ;
 $D^{(0)} \leftarrow W$ ;
for  $K \leftarrow 1$  to  $N$  do
   $D_{K,K}^{(K \cdot b)} \leftarrow \text{FLOYD-WARSHALL}(b, D_{K,K}^{(K \cdot b - b)});$ 
  for all  $1 \leq I \leq N (I \neq K)$  do
     $D_{I,K}^{(K \cdot b)} \leftarrow \text{APSP\_MMA}(b, D_{I,K}^{(K \cdot b - b)}, D_{K,K}^{(K \cdot b)}, \bar{O});$ 
  end for
  for all  $1 \leq J \leq N (J \neq K)$  do
     $D_{K,J}^{(K \cdot b)} \leftarrow \text{APSP\_MMA}(b, D_{K,K}^{(K \cdot b)}, D_{K,J}^{(K \cdot b - b)}, \bar{O});$ 
  end for
  for all  $1 \leq I, J \leq N (I \neq K \text{ and } J \neq K)$  do
     $D_{I,J}^{(K \cdot b)} \leftarrow \text{APSP\_MMA}(b, D_{I,K}^{(K \cdot b)}, D_{K,J}^{(K \cdot b)}, D_{I,J}^{(K \cdot b - b)});$ 
  end for
end for
return  $D^{(n)}$ ;

```

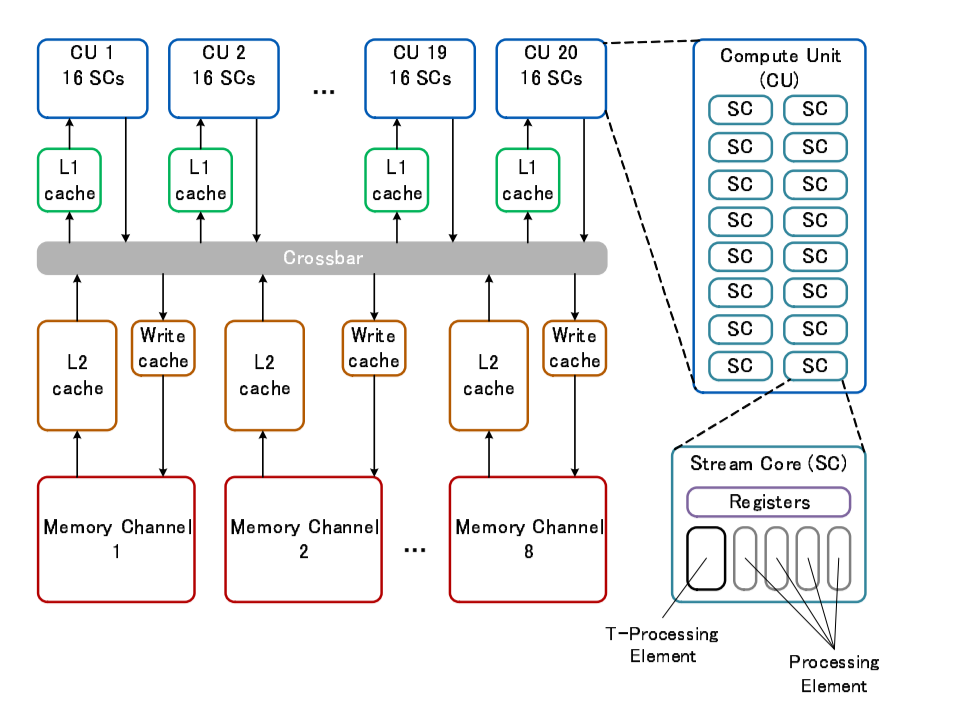


Matrix-Matrix Multiplication for APSP Problem

- Matrix multiply-add in linear algebra
 - $C \leftarrow C + A \times B$
 - $c(i, j) \leftarrow c(i, j) + \sum_{k=1}^n a(i, k) \cdot b(k, j)$
- Matrix multiply-add in min-plus algebra
 - Replace $+$ by \min and \times by $+$.
 - $C \leftarrow C \oplus A \otimes B$
 - $c(i, j) \leftarrow \min(c(i, j), \min_{k=1}^n (a(i, k) + b(k, j)))$

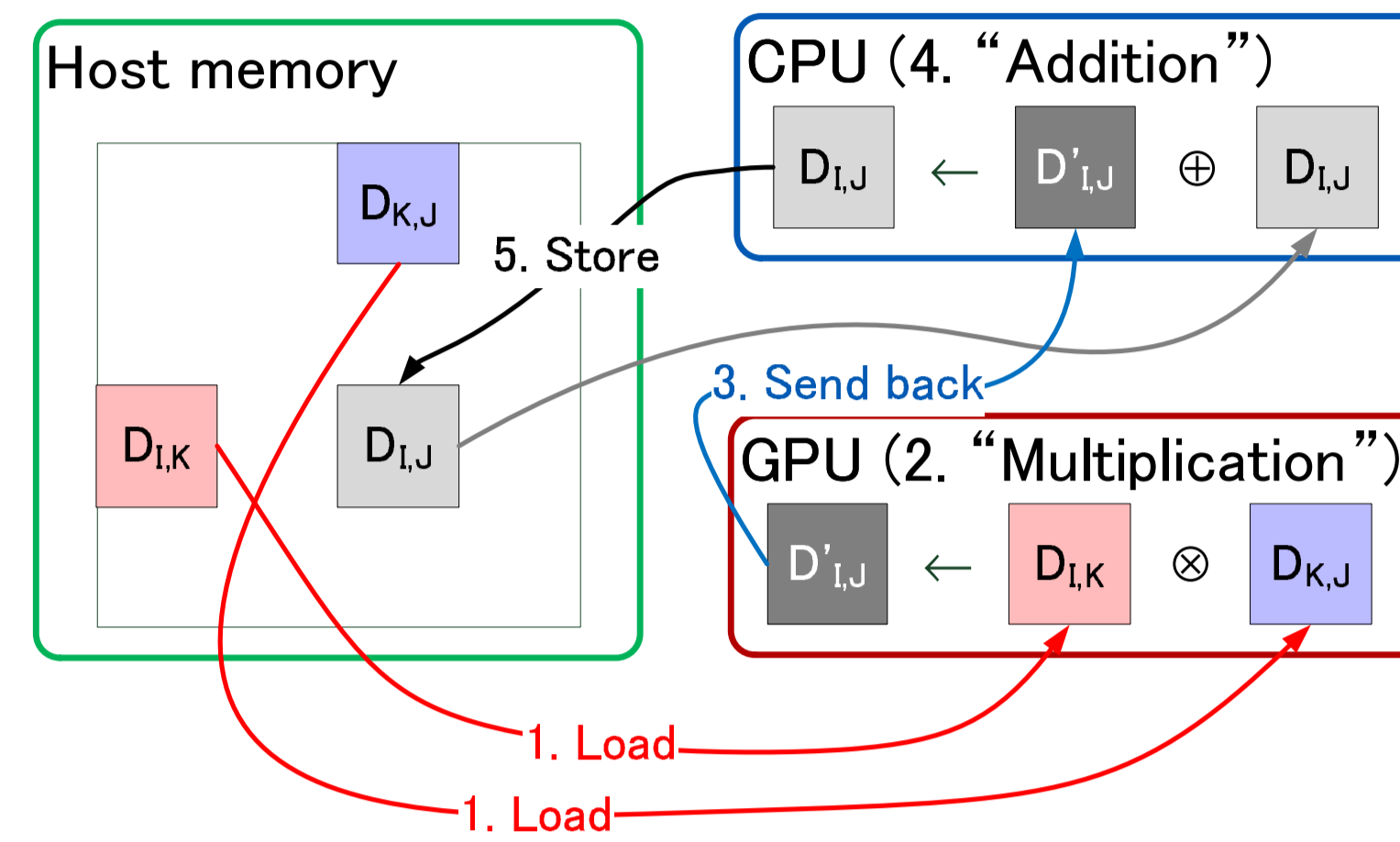
AMD Cypress GPU (Radeon HD 5870)

- 1600 single-precision (SP) floating-point units
- 3200 SP flops/clock by using MAD (fused multiply-add) instructions
- SP peak perf.: 2720 GFlop/s (= 3200 [flops/clock] · 0.85 [GHz])
- Memory bandwidth:
 - L1 cache read: 1088 GB/s
 - L2 cache read: 435 GB/s
 - Global memory: 154 GB/s



Separating Matrix Operations

In our matrix "multiply-add" implementation, the matrix "multiplication" (arithmetic addition) and matrix "addition" (minimum) operations are separated.



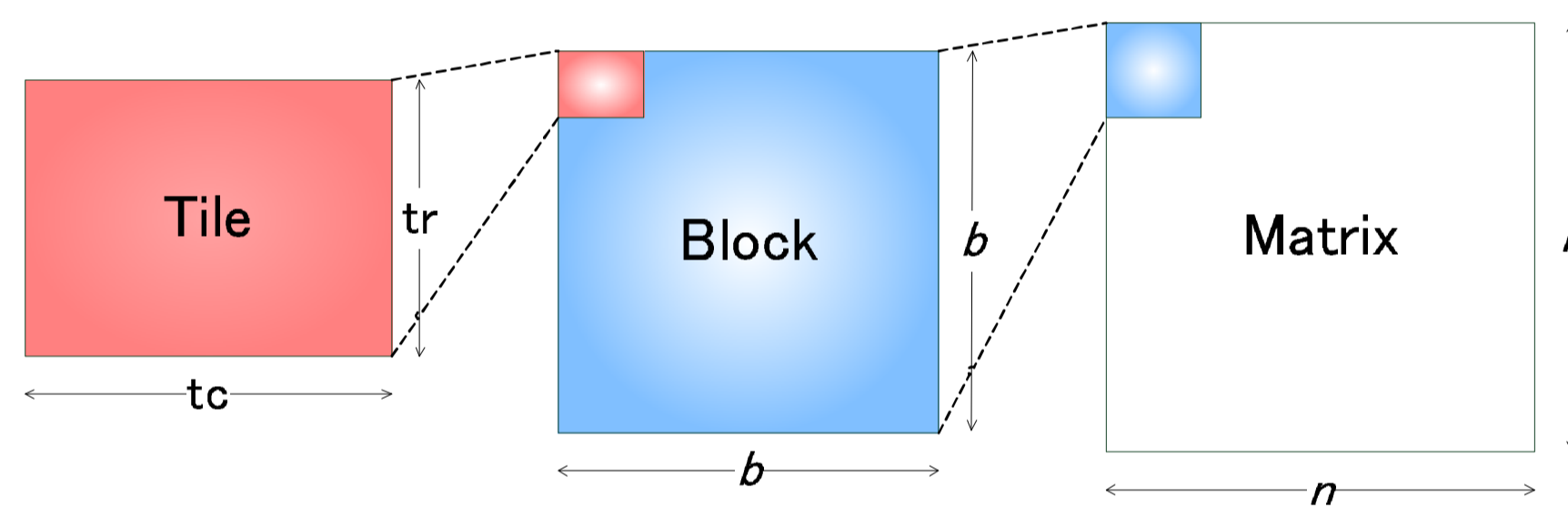
Matrix "Multiplication" Kernel on GPU

Matrix-Matrix Multiplication Kernel

- SGEMM (Single-precision General Matrix-Matrix Multiply) kernel for the AMD Cypress GPU
- Ref.: N. Nakasato, *A fast GEMM implementation on the Cypress GPU*, ACM SIGMETRICS Performance Evaluation Review, vol. 38, no. 4, pp. 50-55, Mar. 2011.
- The kernel was written in assembler-like language called *Intermediate Language (IL)*.
- Max perf.: 2137 GFlop/s (79% SP efficiency)
- Performance comparison:
 - 660 GFlop/s (65% efficiency) on an NVIDIA Fermi GPU (Tesla C2050) by MAGMA BLAS

Tiling in the Kernel

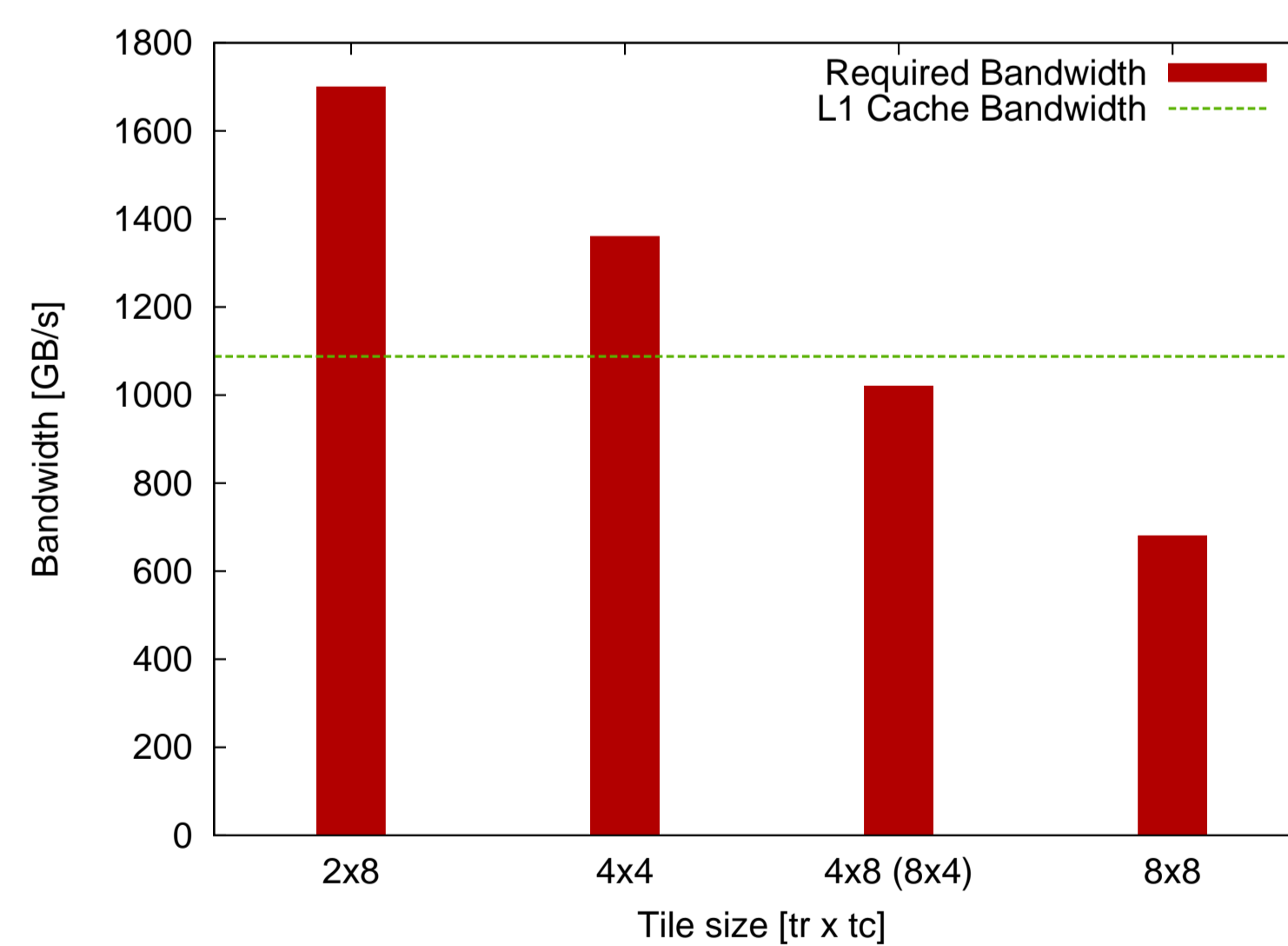
- Further blocking (tiling) in the matrix "multiplication" kernel for the APSP problem
- To alleviate the required memory bandwidth within the kernel.
- Tile size (t_r, t_c) determines the amount of workload of a single thread on the GPU.



Required Bandwidth

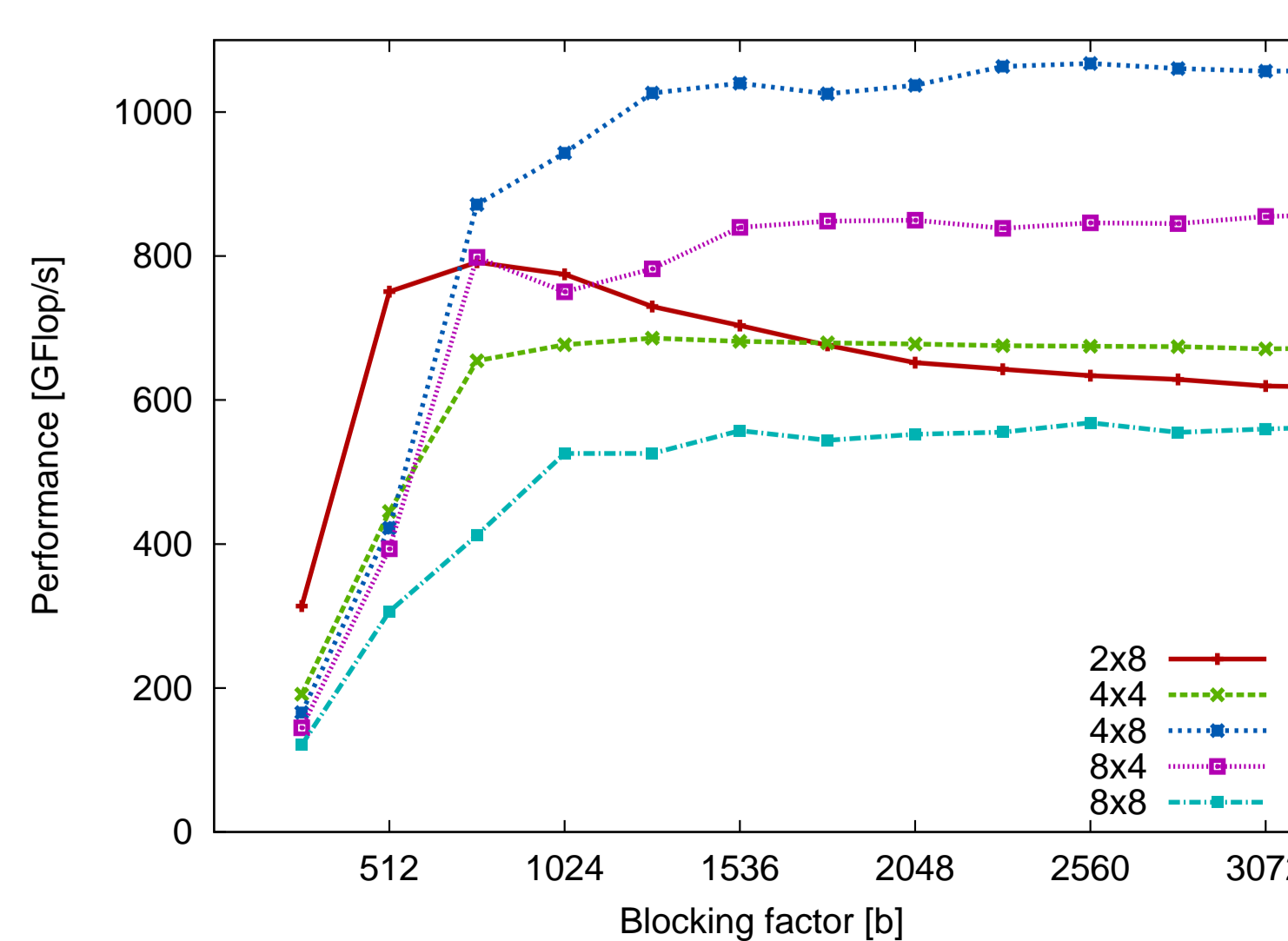
- Required bandwidth (RB_{kernel}) = $\frac{\text{Peak Performance}}{\text{Arithmetic Intensity}}$
- Peak Performance for APSP kernel = 1360 GFlop/s (=2720/2)
- Arithmetic Intensity = $2t_r t_c / (4(t_r + t_c))$
- t_r, t_c are the tile size and 4 means the size of a single-precision floating-point variable in byte.

- When using 4x8 or larger tiles, RB_{kernel} becomes lower than the GPU's L1 cache read bandwidth (1088 GB/s).



Performance of the APSP Kernel

- 4x8 tile is optimal.
 - 1068 GFlop/s (78% efficiency)
- Choosing too large tile size is not good.
 - Not enough number of threads launched.



APSP Implementation on Hybrid CPU-GPU System

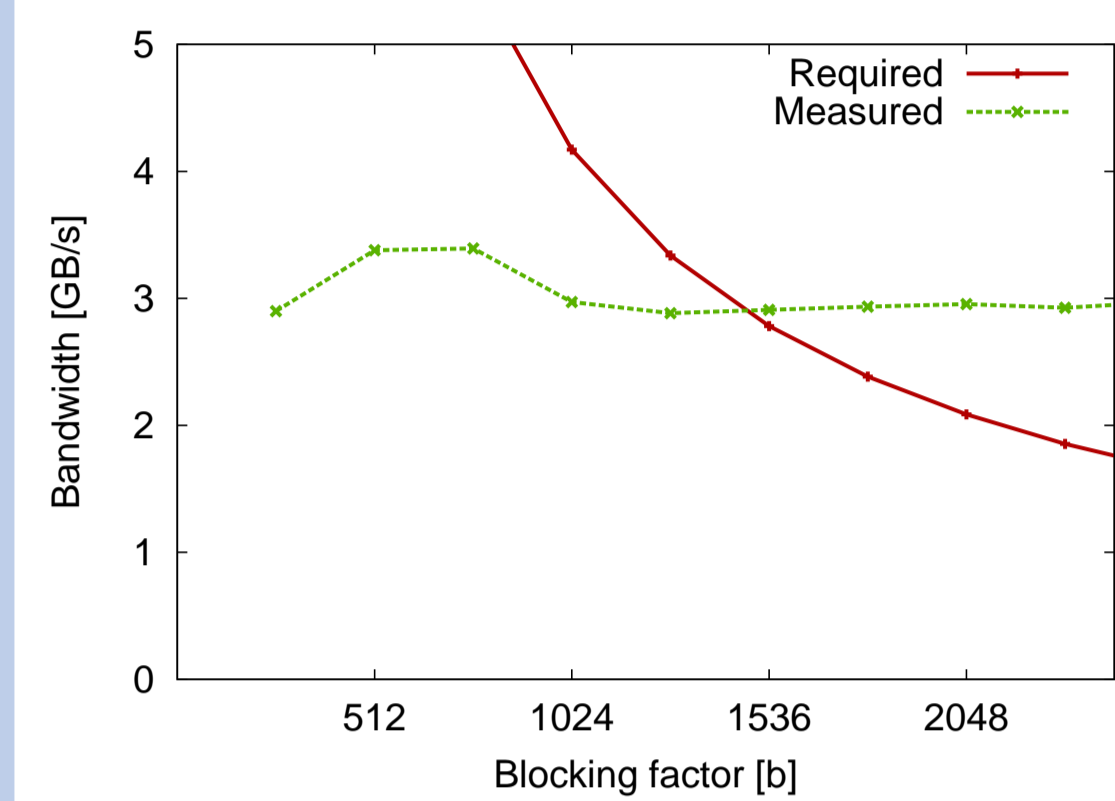
Configuration of the Hybrid System

- AMD Cypress GPU (Radeon HD 5870)
- Intel Gulftown CPU (Core i7 970)
 - 6-core processor at 3.2 GHz
- CPU and GPU are connected through PCI-Express buses
 - Aggregate peak bandwidth: 8 GB/s

Required Bandwidth for Efficient APSP implementation

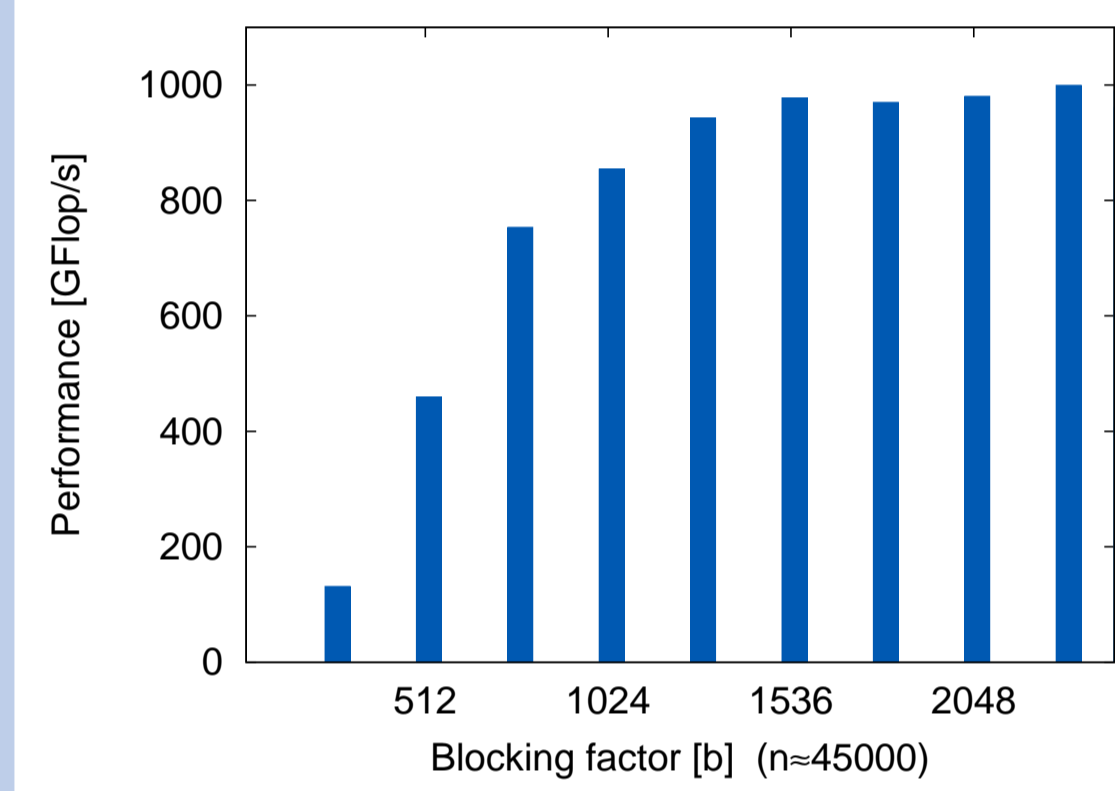
- Required bandwidth (RB_{APSP}) = $\frac{\text{Peak Performance}}{\text{Arithmetic Intensity}}$
- Peak Performance (APSP kernels' perf.) = 1068 GFlop/s
- Arithmetic Intensity = $2b^3 / (4b^2) = b/4$
 - b is the blocking factor, the latter 2 means that 2 blocks of $b \times b$ submatrices are required to send/receive during a single APSP kernel execution, and 4 means the size of a single-precision floating-point variable in byte.

Required and Measured Bandwidth



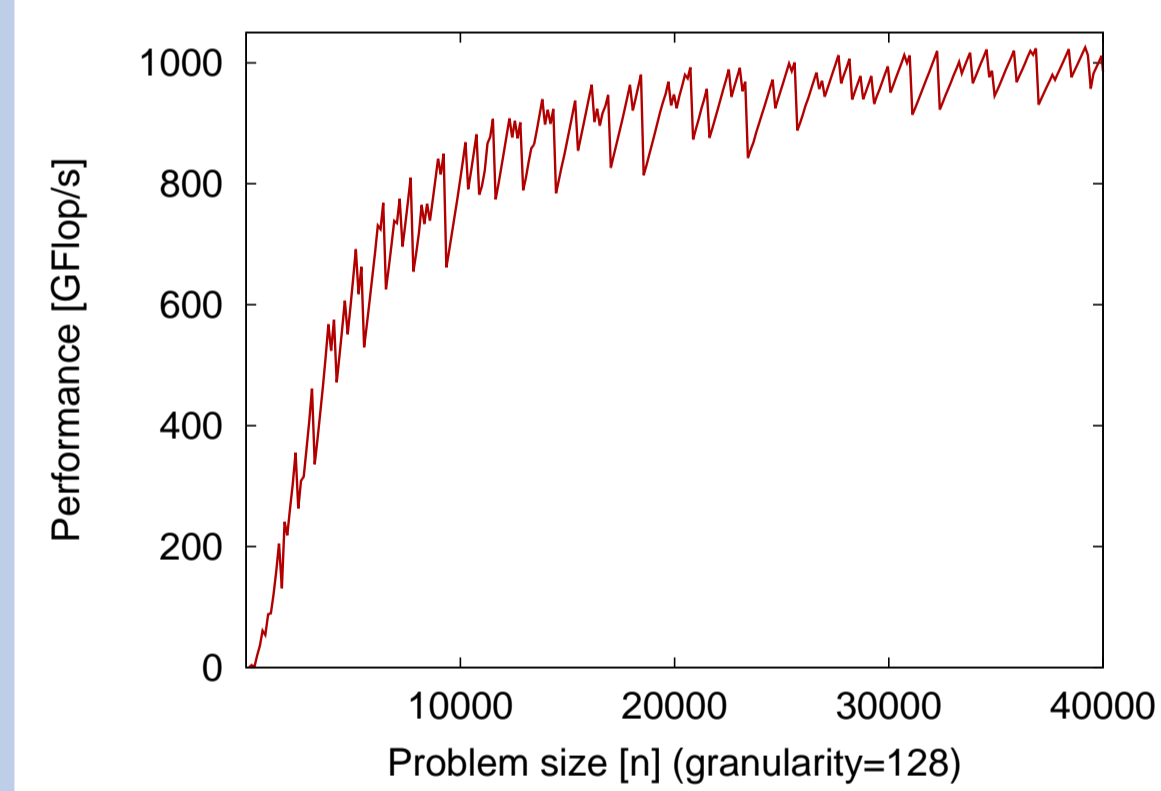
- When using 1536 or larger blocking factor, RB_{APSP} becomes lower than measured bandwidth (about 3 GB/s).

Maximum Performance for Different Blocking Factors



- When the blocking factor b is 1536 or larger, the maximum performance is saturated in around 1 TFlop/s.

Performance of APSP Implementation with Optimal Blocking Factor

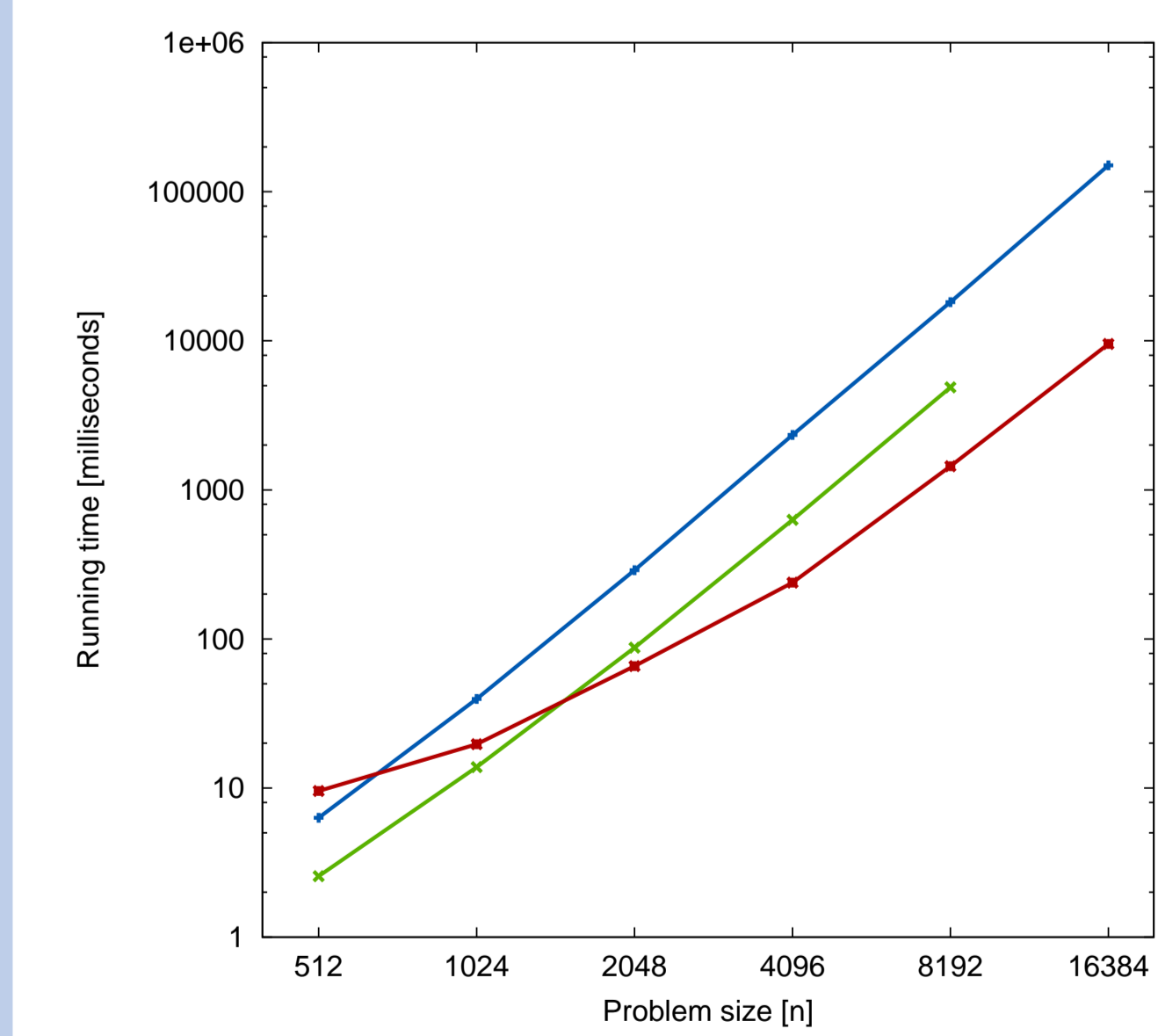


- Choosing an optimal blocking factor can avoid significant performance degradation.
- The optimal blocking factor suited for every problem size is empirically found.

Performance Comparison

Processor Specification

Processor	SP Peak Perf. [GFlop/s]	Clock Speed [GHz]	Num. of Cores
AMD Radeon HD 5870	2720	0.85	1600
NVIDIA GeForce GTX 280	933	1.296	240
Intel Core i7 970	153.6	3.2	6



[Oku] T. Okuyama, F. Ino, K. Hagi-hara, *Fast Blocked Floyd-Warshall Algorithm on the GPU*, IPSJ Trans. Adv. Computing Systems, vol.3, no. 2, pp. 57-66, 2010 [in Japanese].

Conclusion

- Blocked APSP algorithm for hybrid system
 - Matrix "multiplication" is the compute intensive part.
- Implementation is fast when the problem size (number of vertices) is large.
 - Maximum performance ($n = 43776$): 1 TFlop/s
- Future work: blocked APSP algorithm for cluster systems consisting of multiple CPU-GPU nodes.