OpenCLによるCPU/GPU混在システムでの計算高速化の実現に向けて

中里直人 (会津大学)

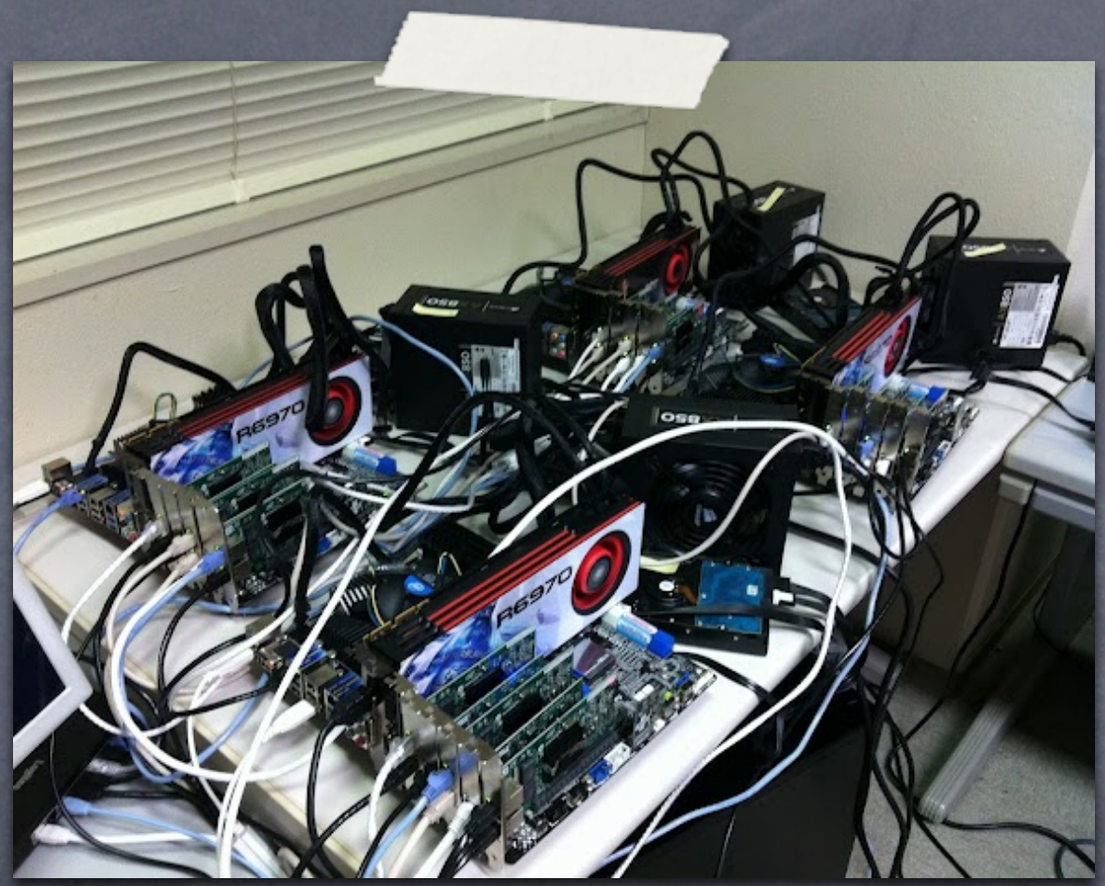
2012年3月6日 九州大学
情報基盤研究開発センター

自己紹介

- 中里直人 (なかさと なおひと)
  - 会津大学 准教授 博士(理学)
  - 専門分野 天文シミュレーションとHPC
    - SPH法による銀河形成シミュレーション
    - FPGAボードによる高速計算
    - 専用計算機の研究開発
    - 専用計算機用ソフトウエアの研究開発
    - メニーコア計算機(GRU,GRAPE-DR)による高性能計算
  - http://galaxy.u-aizu.ac.jp/trac/note/

Agenda

- CPU/GPU混在システムの現状
- AMD GPUによる結果の紹介
- 総合的な並列プログラミング手法としてのOpenCL
- 高速計算とメタプログラミング



GPU/CPU混在システムについて

CPU/GPU混在システム

- CPU with Accelerators
  - 一部の処理を「accelerator」に割り当てる
  - CPU: 粒度の粗いタスク
    - high performance cores with short vector
    - heavyweight thread ~ 4 - 16 in flight
    - registers 16x4w (x86_64), 128x2w (VENUS)
  - GPU: 粒度の細かいタスク
    - low performance cores (ALUs) : scaler or SIMD/VLIW
    - lightweight threads ~ 500 - 2000 in flight
    - high bandwidth memory but large latency
    - registers per thread 128x4w (Cypress,Cayman), 128w (Tahiti)

TOP500 (2011年11月)

1	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect / 2011 Fujitsu	705024	10510.00	11280.38	12659.9
2	National Supercomputing Center in Tianjin China	NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050 / 2010 NUDT	186368	2566.00	4701.00	4040.0
3	DOE/SC/Oak Ridge National Laboratory United States	Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 Cray Inc.	224162	1759.00	2331.00	6950.0
4	National Supercomputing Centre in Shenzhen (NSCS) China	Dawning TC3600 Blade System, Xeon X5650 6C 2.66GHz, Infiniband QDR, NVIDIA 2050 / 2010 Dawning	120640	1271.00	2984.30	2580.0
5	GSIC Center, Tokyo Institute of Technology Japan	HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows / 2010 NEC/HP	73278	1192.00	2287.63	1398.6
6	DOE/NNSA/LANL/SNL United States	Cray XE6, Opteron 6136 8C 2.40GHz, Custom / 2011 Cray Inc.	142272	1110.00	1365.81	3980.0
7	NASA/Ames Research Center/NAS United States	SGI Altix ICE 8200EX/8400EX, Xeon HT QC 3.0/Xeon 5570/5670 2.93 Ghz, Infiniband / 2011 SGI	111104	1088.00	1315.33	4102.0
8	DOE/SC/LBNL/NERSC United States	Cray XE6, Opteron 6172 12C 2.10GHz, Custom / 2010 Cray Inc.	153408	1054.00	1288.63	2910.0
9	Commissariat a l'Energie Atomique (CEA) France	Bull bullx super-node S6010/S6030 / 2010 Bull	138368	1050.00	1254.55	4590.0
10	DOE/NNSA/LANL United States	BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband / 2009 IBM	122400	1042.00	1375.78	2345.0

GPU 2.5

GPU 1.3

GPU 1.2

CellBE 1.0

Green 500 (2011年11月)

Green500 Rank	MFLOPS/W	Site*	Computer*	Total Power (kW)
1	2026.48	IBM - Rochester	BlueGene/Q, Power BQC 16C 1.60 GHz, Custom	85.12
2	2026.48	IBM Thomas J. Watson Research Center	BlueGene/Q, Power BQC 16C 1.60 GHz, Custom	85.12
3	1996.09	IBM - Rochester	BlueGene/Q, Power BQC 16C 1.60 GHz, Custom	170.25
4	1988.56	DOE/NNSA/LLNL	BlueGene/Q, Power BQC 16C 1.60 GHz, Custom	340.50
5	1689.86	IBM Thomas J. Watson Research Center	NNSA/SC Blue Gene/Q Prototype 1	38.67
6	1378.32	Nagasaki University	DEGIMA Clu adeon GPU, Infiniband QDR	47.05
7	1266.26	Barcelona Supercomputing Center	Bu B505, Xeon E5649 6C 2.53GHz, Infiniband QDR, NV DIA 2090	81.50
8	1010.11	TGCC / GENCI	Cu e Hybrid Nodes - Bullx B505, Nvidia M2090, Xeon E5 40 2	108.80
9	963.70	Institute of Process Engineering, Chinese Academy of Sciences	Mo e-8.5 .27 GHz, Infiniband QD R, NVIDIA 2050	515.20
10	958.35	GSIC Center, Tokyo Institute of Technology	HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Li x/Windows	1243.80

AMD

NVIDIA

GPUによるHPLについて

- DGEMMの部分をGPUで計算
  - GPUの性能に強く依存する
  - NVIDIAのシステム R_{peak}/R_{max} ~ 54.5%

2	National Supercomputing Center in Tianjin China	NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050 / 2010 NUDT	186368	2566.00	4701.00	4040.0

- AMDのシステム　R_{peak}/R_{max} ~ 58% (1 node)

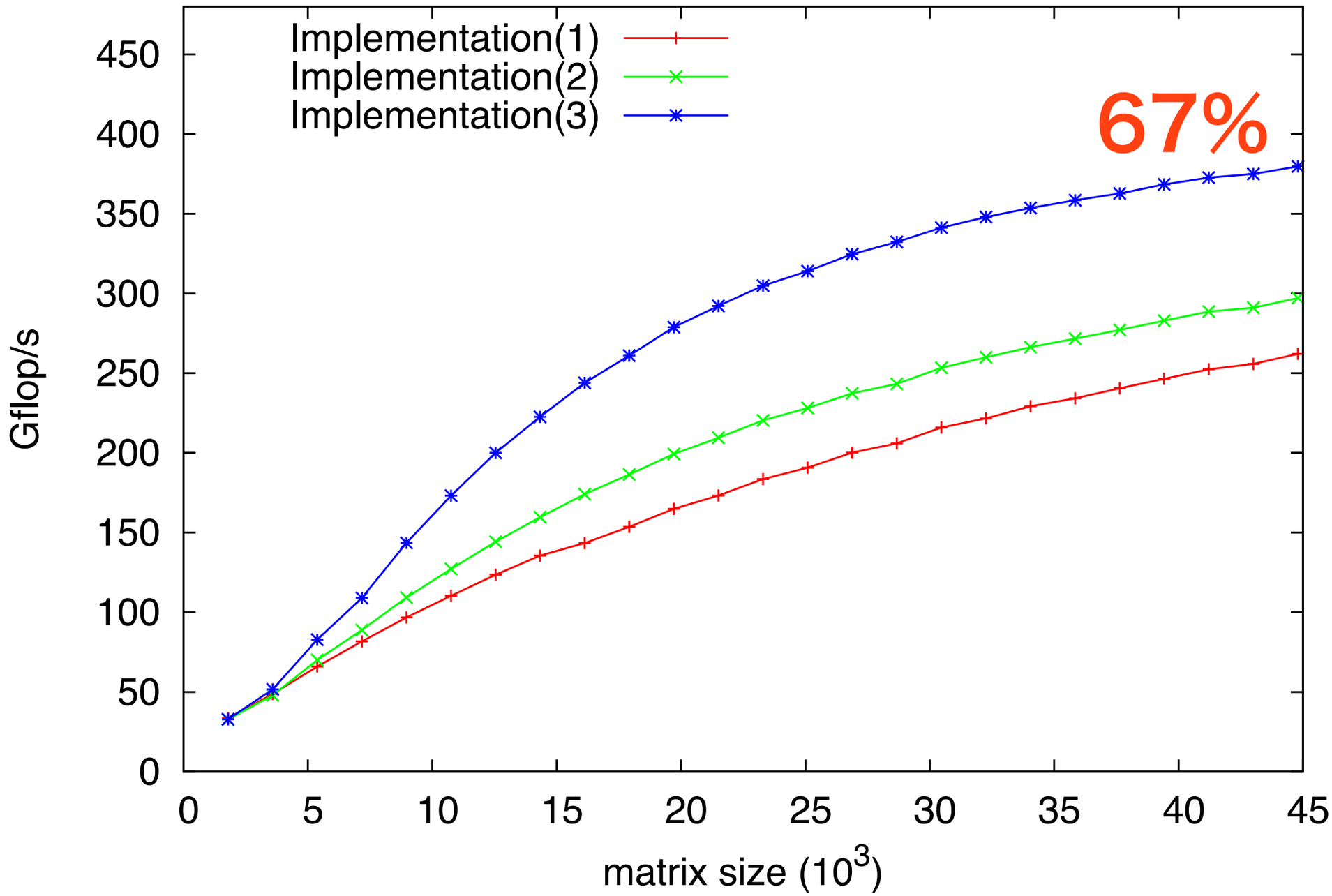
33	Universitaet Frankfurt Germany	Supermicro Cluster, QC Opteron 2.1 GHz, ATI Radeon GPU, Infiniband / 2011 Clustervision/Supermicro	16368	299.30	508.50	416.8

1 node 70%

http://dx.doi.org/10.1007/s00450-011-0161-5

AMD GPUでのLU分解

- 酒井,松本,中里&Sedukhin (会津大学)
  - 第73回全国大会講演論文集 Vol. 1, 205 - 207 (2011)

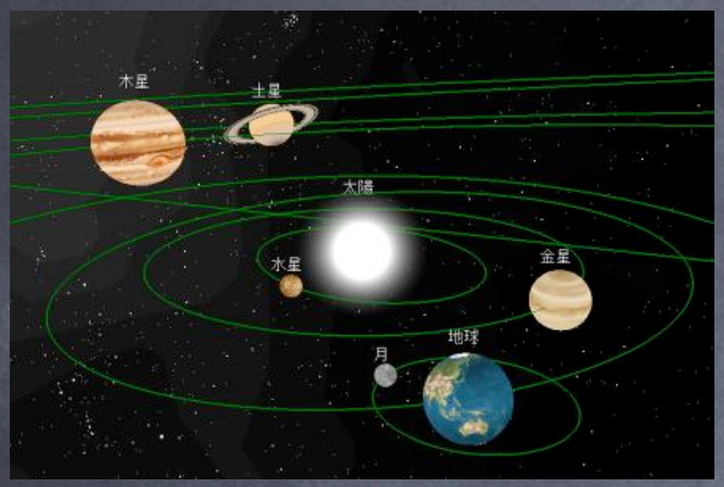


HPL/LU分解のポイント

- CPUとGPUをどちらも活用する
  - R_{max}は両者の性能の和になっている
  - CPUとGPUの性能比により最適化手法が異なる
    - 天河-1　　CPU 140 + GPU 515 GF　　(ノードあたり GFLOPS)
    - TSUBAME2　CPU 140 + GPU 1545 GF
    - Frankfurt　　CPU 202 + GPU 544 GF
- R_{peak}/R_{max} はメインメモリ量に比例
  - ホストのメモリが多い方が性能高い
    - ブロッキングのためGPUのメモリ量は重要ではない
  - PCIeとインターコネクトの転送性能も重要ではない

粒子シミュレーション in 宇宙

solar system

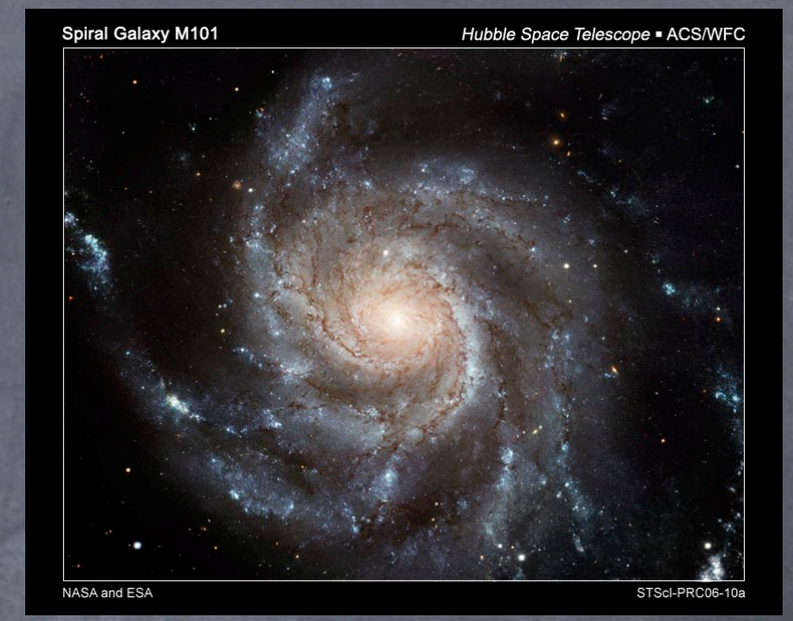


$$N \sim 10$$
$$t_{\rm lifetime} \sim 10^9 {\rm yr}$$

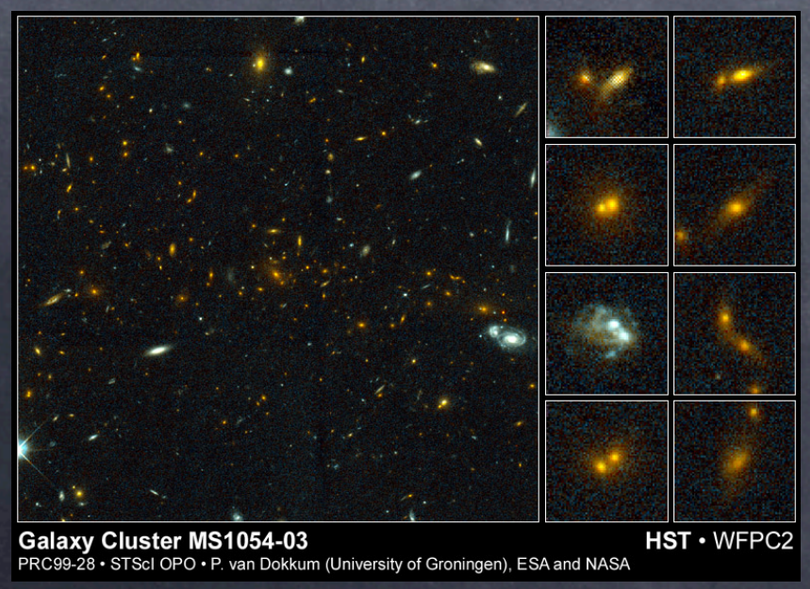
star cluster

$$N \sim 10^5$$
$$t_{\rm lifetime} \sim 10^{10} {\rm yr}$$

galaxy



$$N \sim 10^{11}$$
$$t_{\rm lifetime} \sim 10^{10} {\rm yr}$$



cluster of galaxies

$$N \sim 10^3$$
$$t_{\rm lifetime} \sim 10^{10} {\rm yr}$$

Numerical model

solar system

sun&planets

$N \sim 10$

$t_{\rm lifetime} \sim 10^{10}{\rm yr}$

$t_{\rm dynamical} \sim 1{\rm yr}$

galaxy

blob of stars&DM

$N \sim 10^6 - 10^7$

$t_{\rm lifetime} \sim 10^{10}{\rm yr}$

$t_{\rm dynamical} \sim 10^8{\rm yr}$

star cluster

individual stars

$N \sim 10^5$

$t_{\rm lifetime} \sim 10^{10}{\rm yr}$

$t_{\rm dynamical} \sim 10^5{\rm yr}$

whole universe

blob of DM

$N \sim 10^9 - 10^{11}$

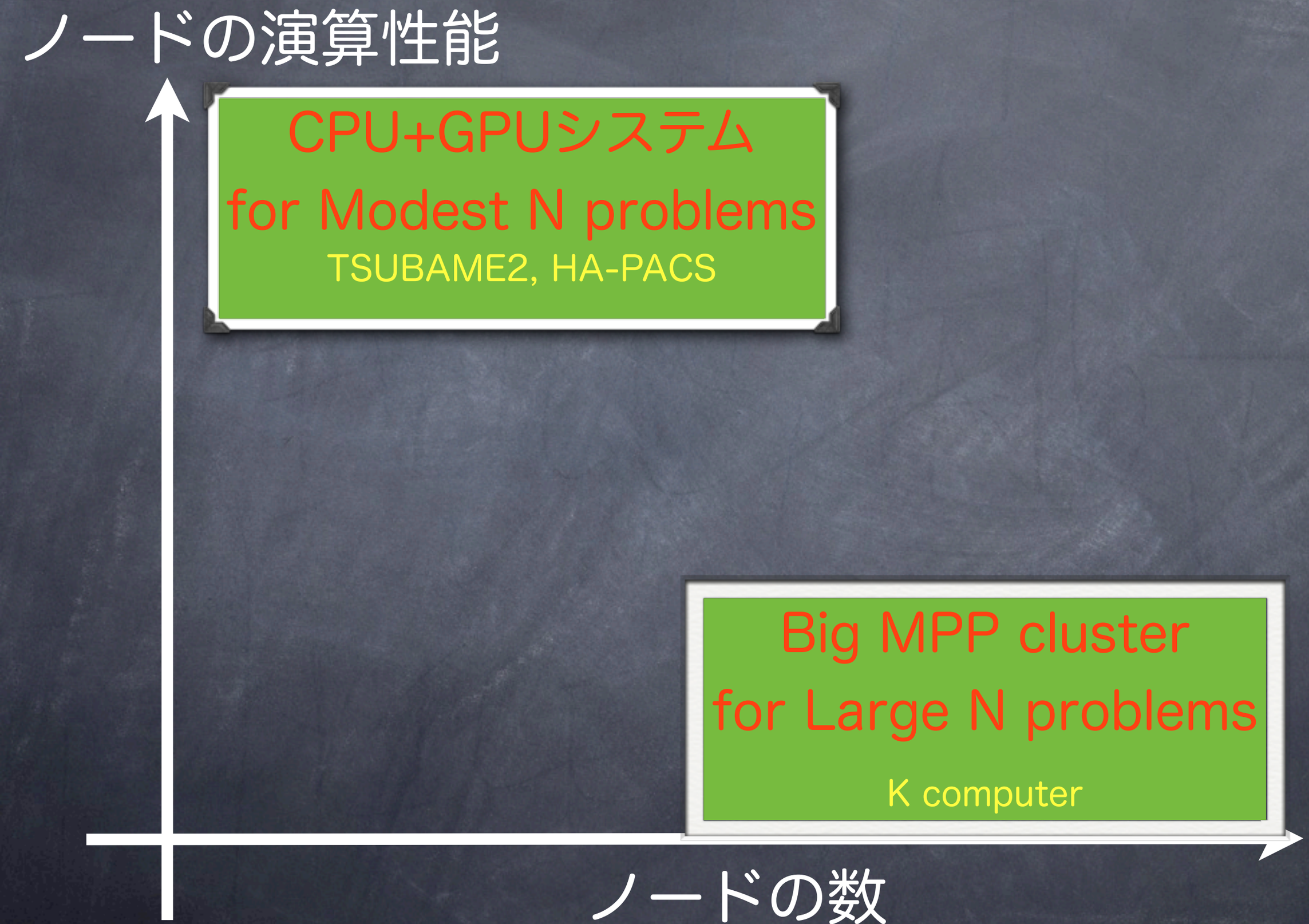
$t_{\rm lifetime} \sim 10^{10}{\rm yr}$

$t_{\rm dynamical} \sim 10^8{\rm yr}$

Grand Challenge Problems

- Simulations with very huge N
  - How is mass distributed in the Universe?
  - Scalable on a simple big MPP system
    - One big run with N ~ $10^{9\text{-}12}$
    - Limited by memory size
- Modest N but complex physics
  - Precise modeling of formation of astronomical objects like galaxy, star, solar system.
  - Need many runs with N ~ $10^{6\text{-}7}$

HPC Cluster Configuration



粒子シミュレーションでの利点

- 天文の粒子シミュレーションの現状
  - MPP向けのコード(Gadget-2)が広く使われている
    - あまりスケーラブルではない
- GPUがあるとノードの演算性能が上がる
  - 与えられた問題を少ないノード数で実行できる
  - GRAPE clusterで実証済み
    - PPPM and TreePM Methods on GRAPE Systems (Yoshikawa&Fukushige 2005)
  - CPU+GPU向けのアルゴリズムの検討

CPU/GPU混在システムの設計

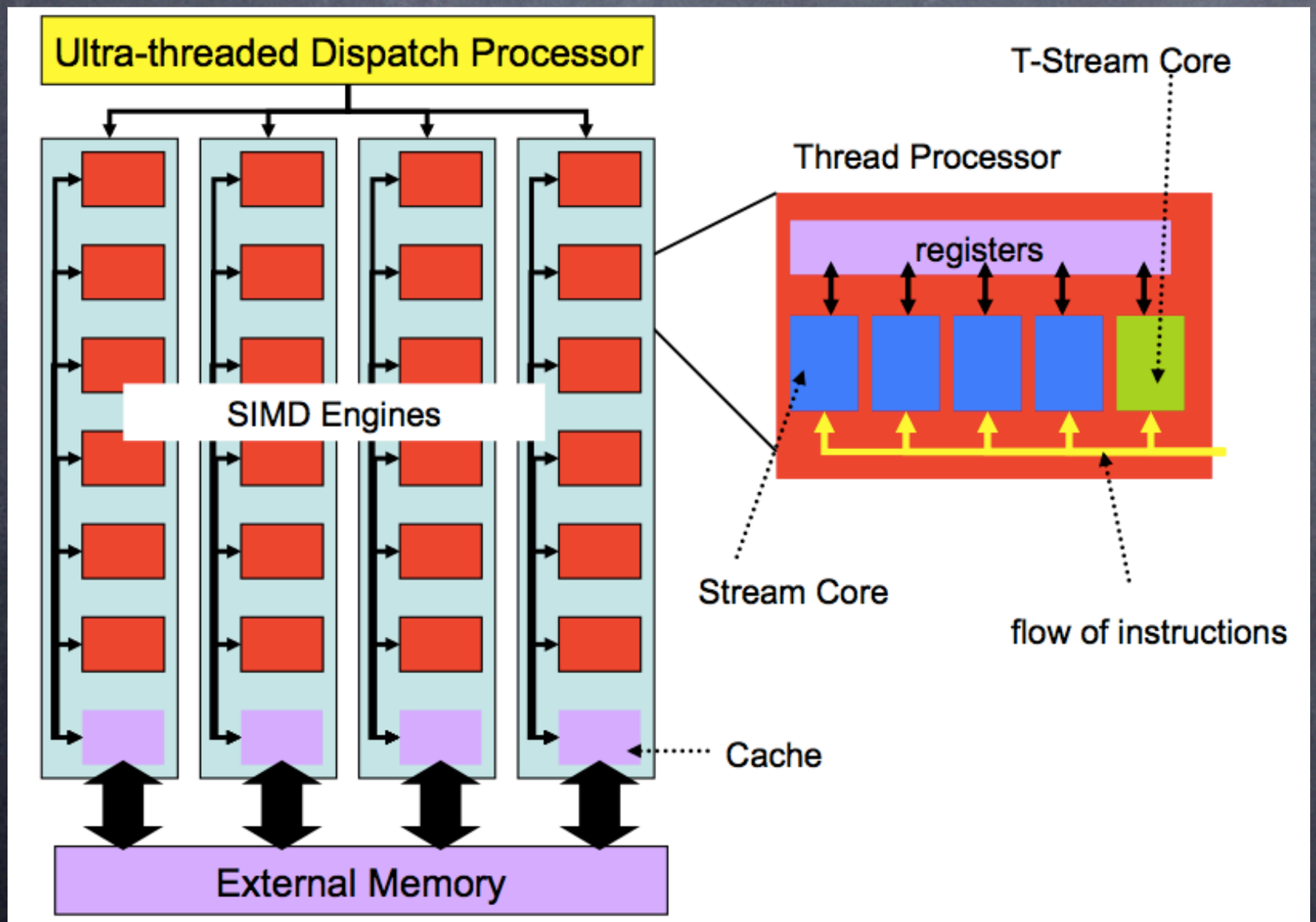
- CPUとGPUの演算性能のバランス
  - 普通のMPPと比べると向き不向きの差が大きい
  - ある種アプリケーション特化システムとして、システムデザインの最適化が必要
  - X CPU + Y GPU + Z Gbps
    - HA-PACSは2 : 4 : 40
- 粒子シミュレーション用なら
  - fatツリーは必ずしも必要ない
  - CPUにも十分な性能が必要
  - GPUとインターコネクトでPCIe帯域を分散

Architecture	Cypress	Fermi
Board Name	Radeon 5870	Tesla C2050
# of SP cores	1600	448
# of DP cores	320	224
# of vector cores	20	14
registers/core	256 KB	128 KB
tex. cache/core	8 KB	12 KB
shared mem./core	32 KB	64 KB
2nd cache	512 KB	768 KB
core clock(GHz)	0.85	1.15
SP peak(Tflop/s)	2.72	1.03
DP peak(Gflop/s)	544	515
memory clock(GHz)	1.2	0.75
memory bus	256	384
memory size(GB)	1	3
memory BW (GB/s)	153.6	144
tex. cache BW(GB/s)	54.5	

GPUアーキテクチャの比較

AMD GPUのアーキテクチャ(1)

- - 2011: RV770, Cypress, Cayman
  - グラフィック処理に最適化されたアーキテクチャ
  - Thread Processor 5 or 4 レーンのVLIW ALU



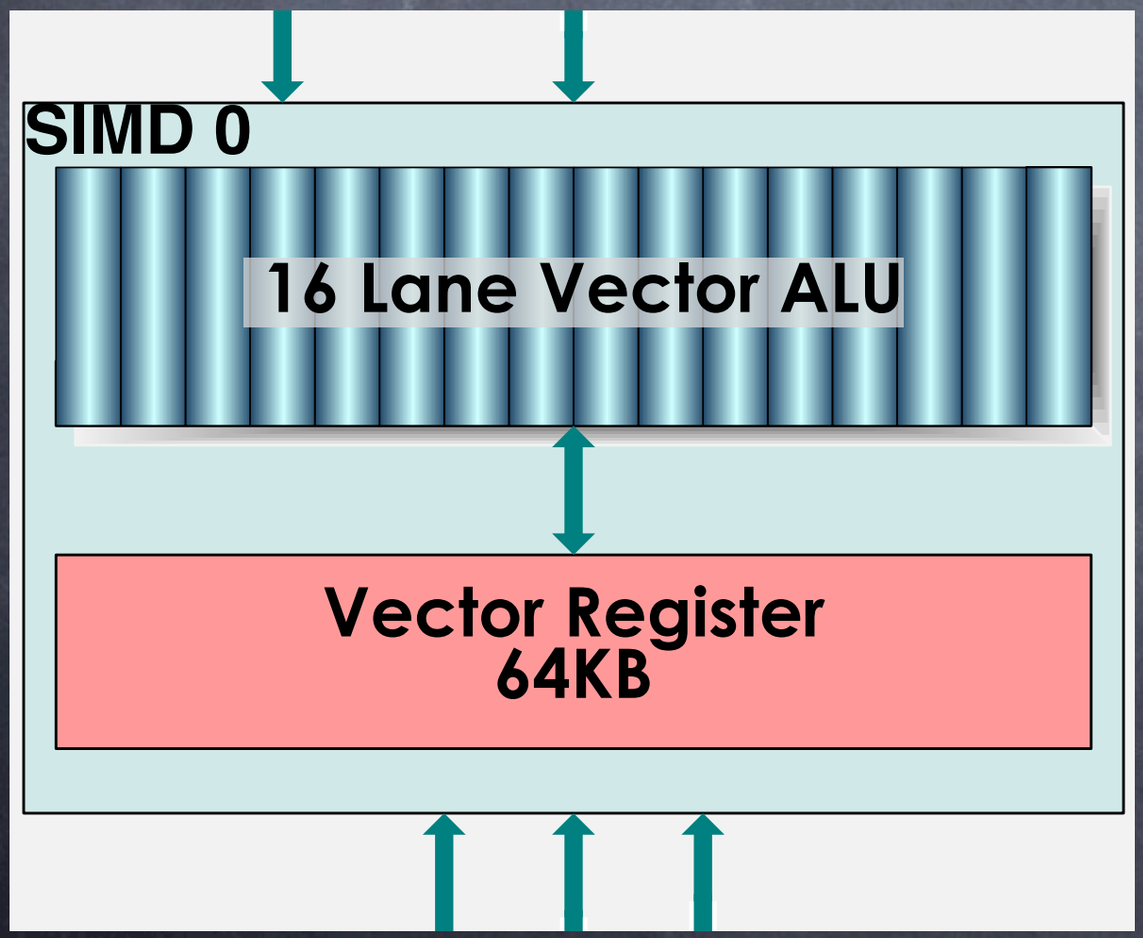
5(4)個のSP FMA
1個のDP FMA
~ 1600 TP

Cypress
~ 2.7 / 0.544 Tflops
Cayman
~ 2.7/0.676 Tflops

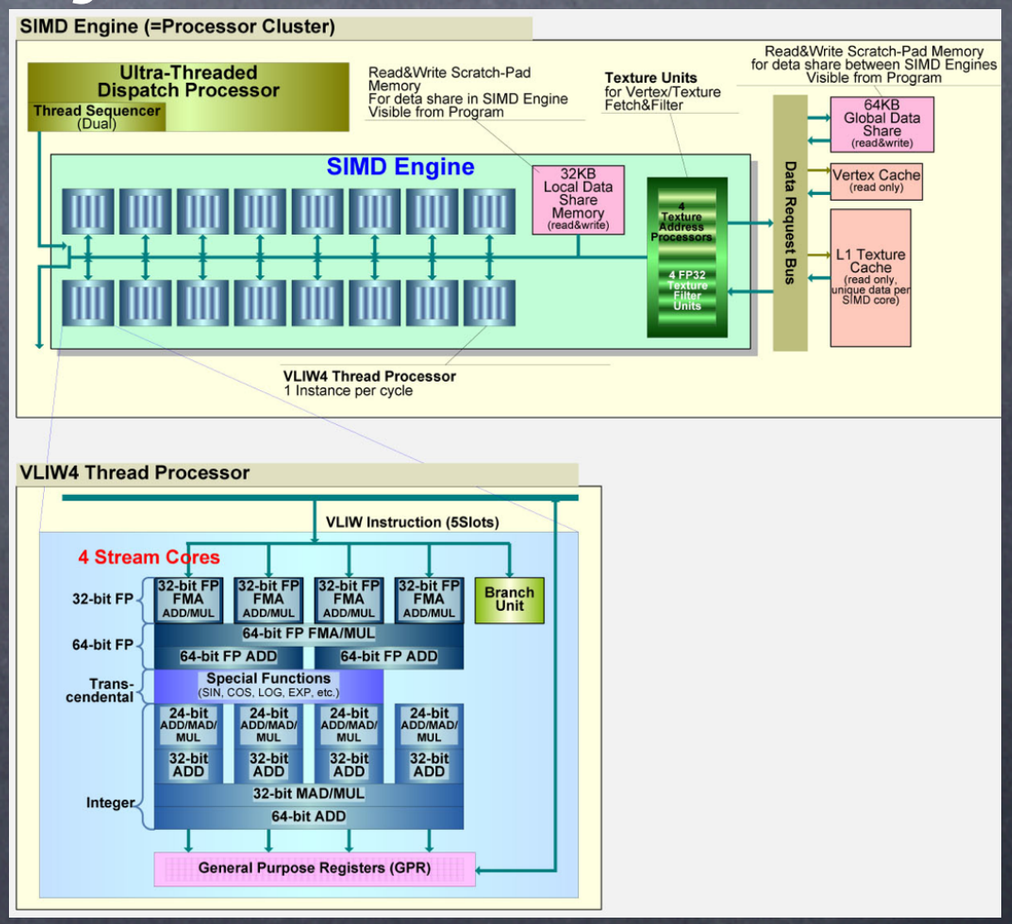
AMD GPUのアーキテクチャ(2)

- 2012: Tahiti
  - VLIW4 から Vector-Scalarの構成へ
  - 3.78 (SP) / 0.95 (DP) TFLOPS

Tahiti



Cayman



http://pc.watch.impress.co.jp/docs/column/kaigai/20120130_507882.html

AMDとNVIDIAのGPUの比較

- Tahiti以前 (-2011)
  - AMD　：グラフィック処理向けに最適化 (VLIW)
    - SP : DP = 5/4 : 1,　分岐粒度 64(80) threads
    - ピーク性能高い, レジスタファイルが多い, Brook+/IL/OpenCL
  - NVIDIA : GPU computingも考慮した最適化
    - SP : DP = 2 : 1,　　分岐粒度 16 threads
    - 共有メモリが多い, CUDA/OpenCL
- Tahiti, Kepler時代 (2012 -)
  - 両者ともスカラー　SP : DP = 2 : 1
  - ~ 1 TFLOPS @ DP

Cypress vs. Fermi

Architecture	Cypress	Fermi
Board Name	Radeon 5870	Tesla C2050
# of SP cores	1600	448
# of DP cores	320	224
# of vector cores	20	14
registers/core	256 KB	128 KB
tex. cache/core	8 KB	12 KB
shared mem./core	32 KB	64 KB
2nd cache	512 KB	768 KB
core clock(GHz)	0.85	1.15
SP peak(Tflop/s)	2.72	1.03
DP peak(Gflop/s)	544	515
memory clock(GHz)	1.2	0.75
memory bus	256	384
memory size(GB)	1	3
memory BW (GB/s)	153.6	144
tex. cache BW(GB/s)	54.5	

NVIDIA GPUでの最適化指針

- 演算密度を高くする
  - 1ワード当たり読み出しあたりの演算量を多くする
    - ブロッキング
    - ループアンローリング
- coalesce access
  - メインメモリの帯域幅を最大化
- 共有メモリの利用
  - ソフトウエアキャッシュとして
  - メインメモリーの読み出しレイテンシが大きいため

AMD GPUでの最適化指針

- 演算密度を高くする
  - レジスタブロッキングが有効
- read onlyキャッシュの利用
  - OpenCLでのImage bufferを利用すること
    - texture unit経由の読み出し ~ 1 TB/sec
    - 共有メモリを利用するまでもない
- ILでのプログラミング
  - ILは仮想的なアセンブリ言語 (レジスタ数無限)
  - 低レベルだが演算器を最大限活用可能

AMD GPUでの研究成果

GRAPE like N-Body $O(N^2)$

- Nakasato 2008-2011: read only キャッシュを利用

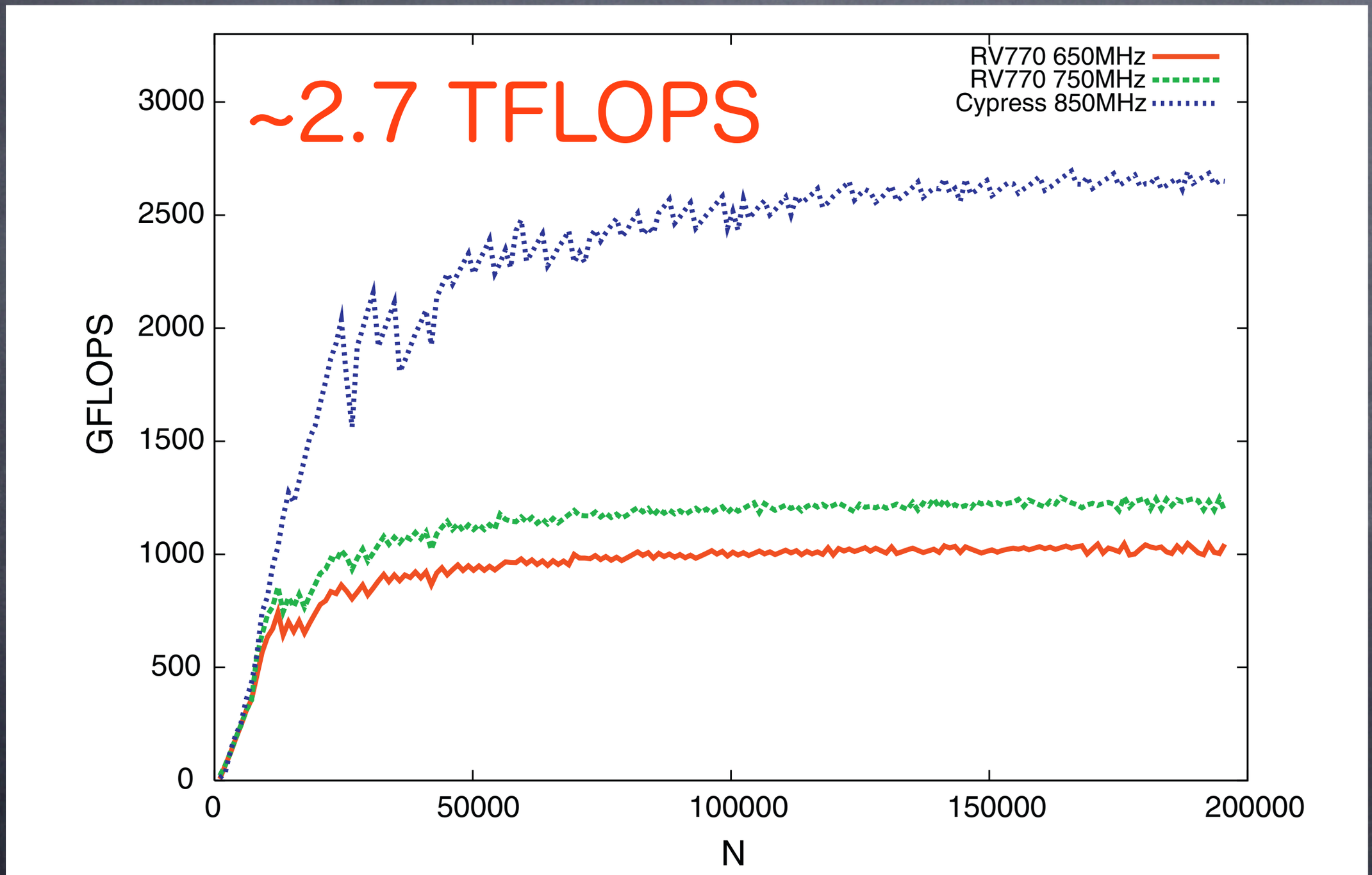
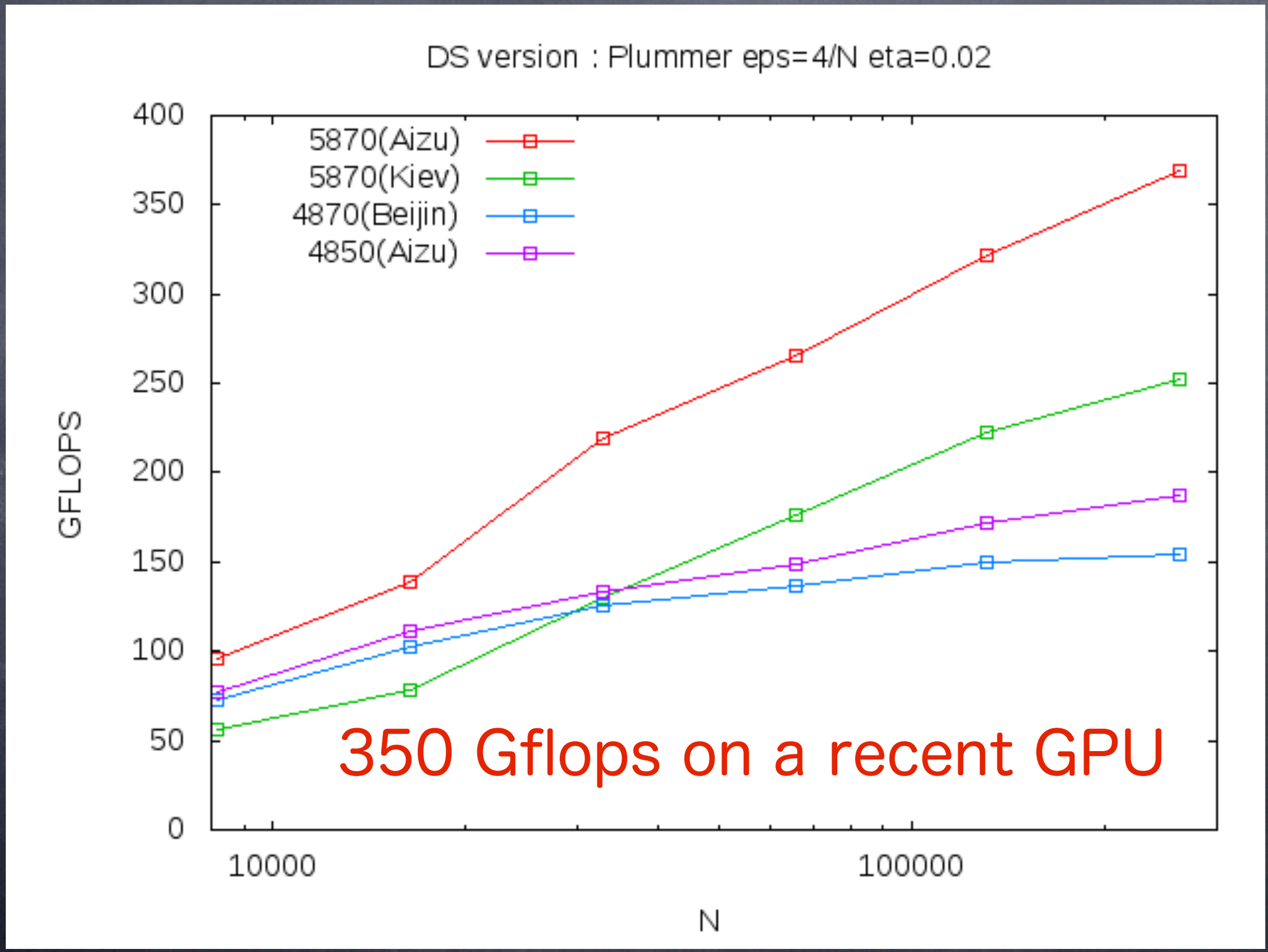
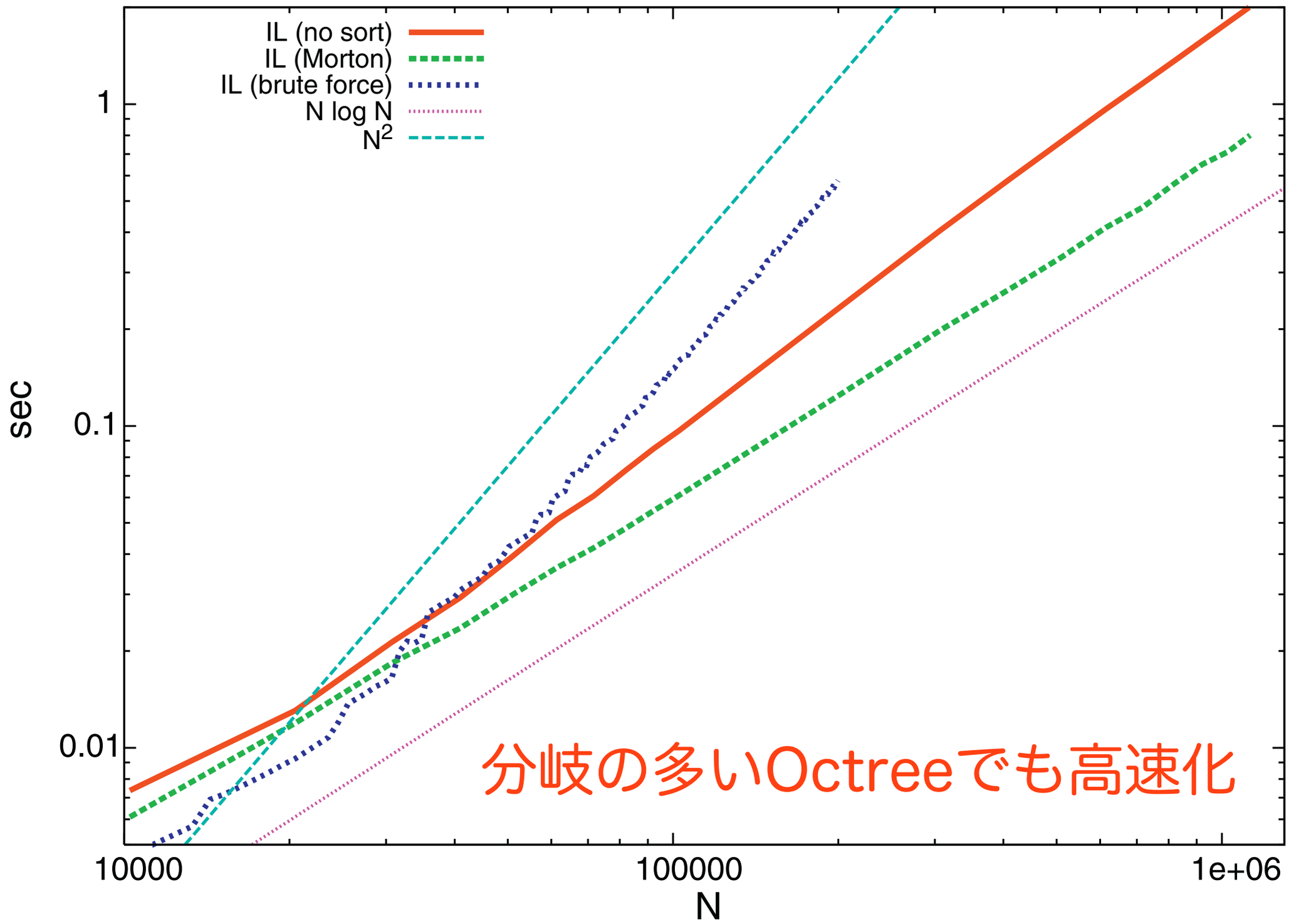


Fig. 2. Performance of the brute force method on various GPUs.

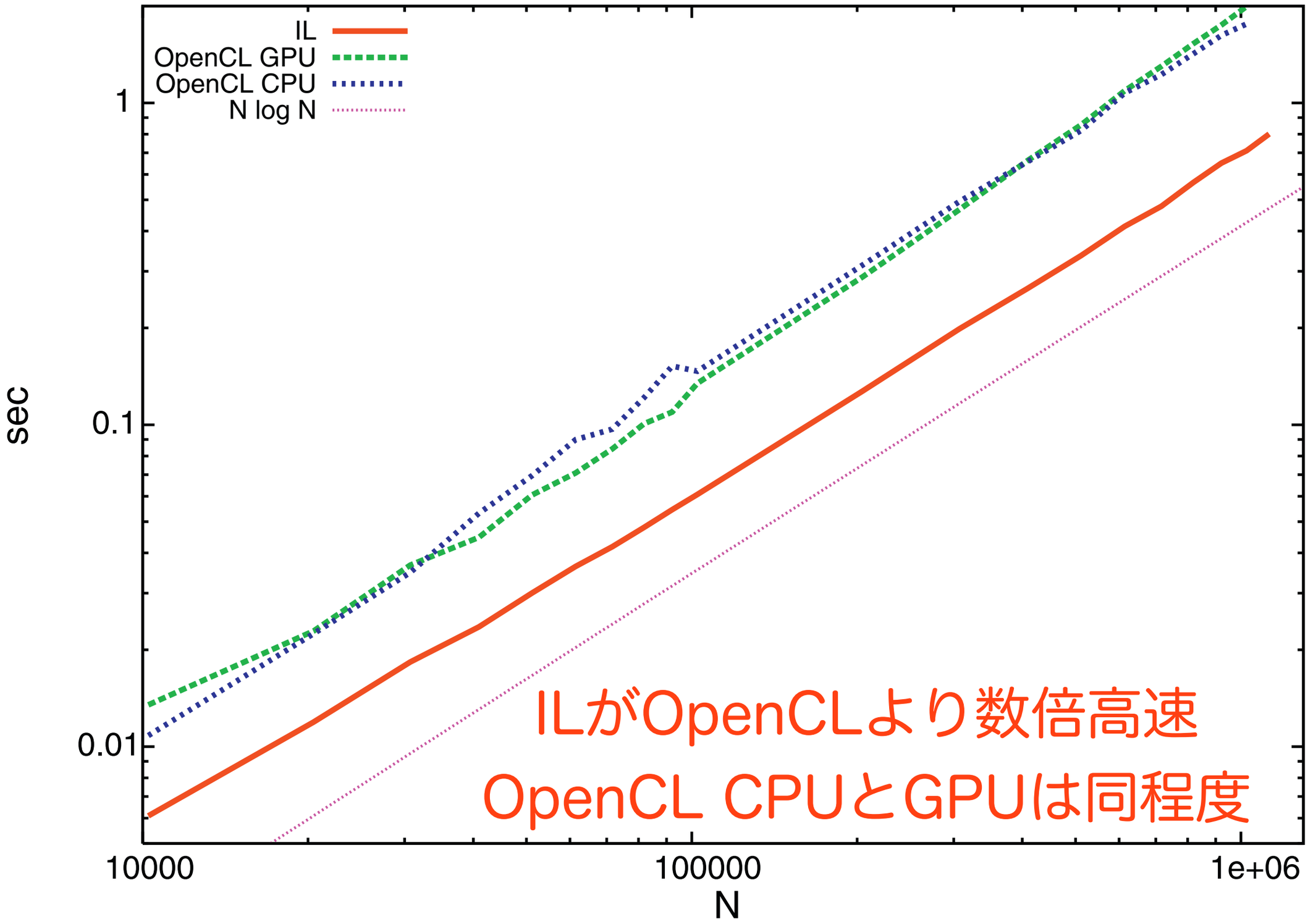
GRAPE-6 互換ライブラリ



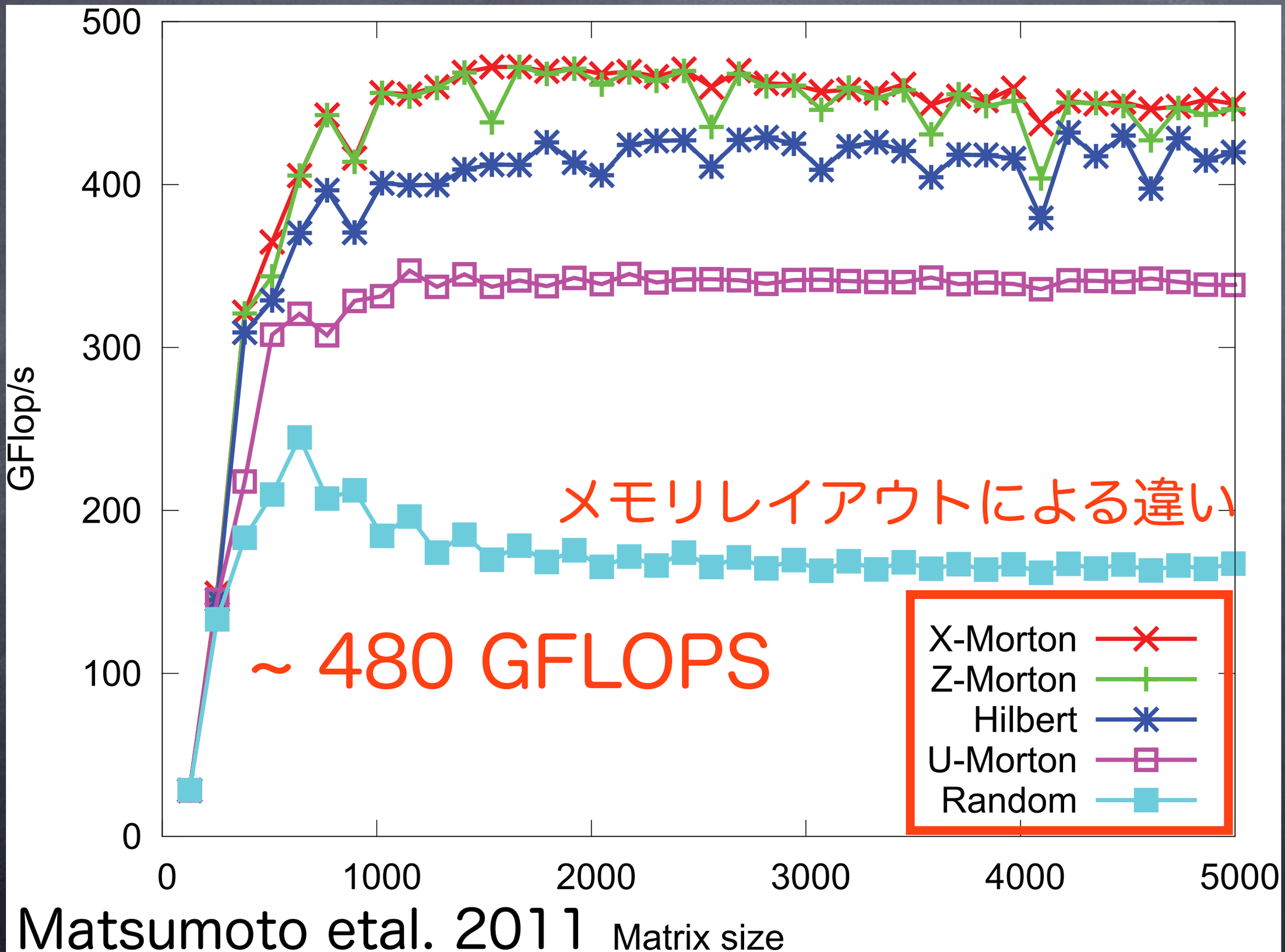
Octree method : O(N log N)



Octree IL vs. OpenCL (2010)

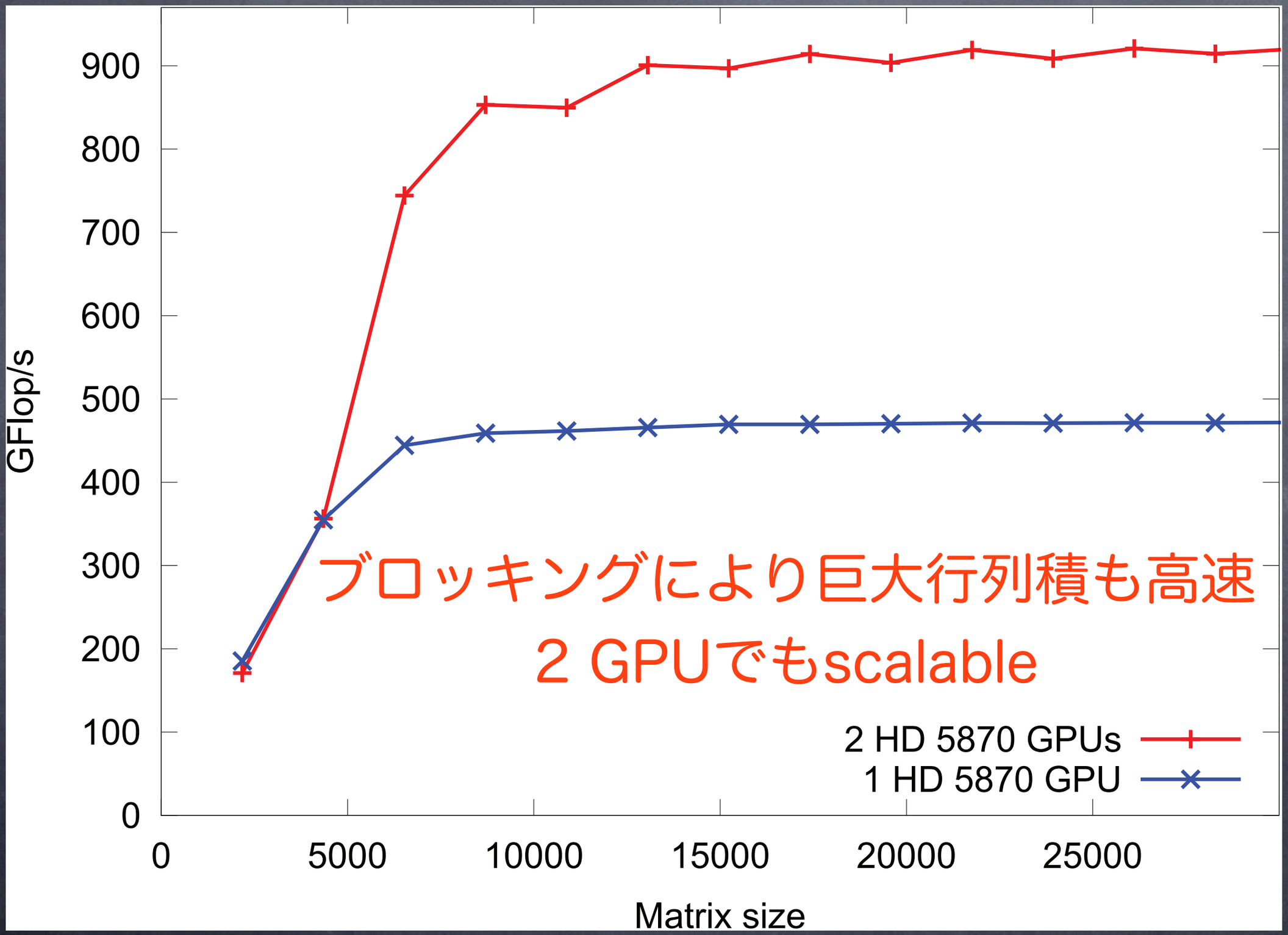


DGEMMカーネルの性能 (2010)



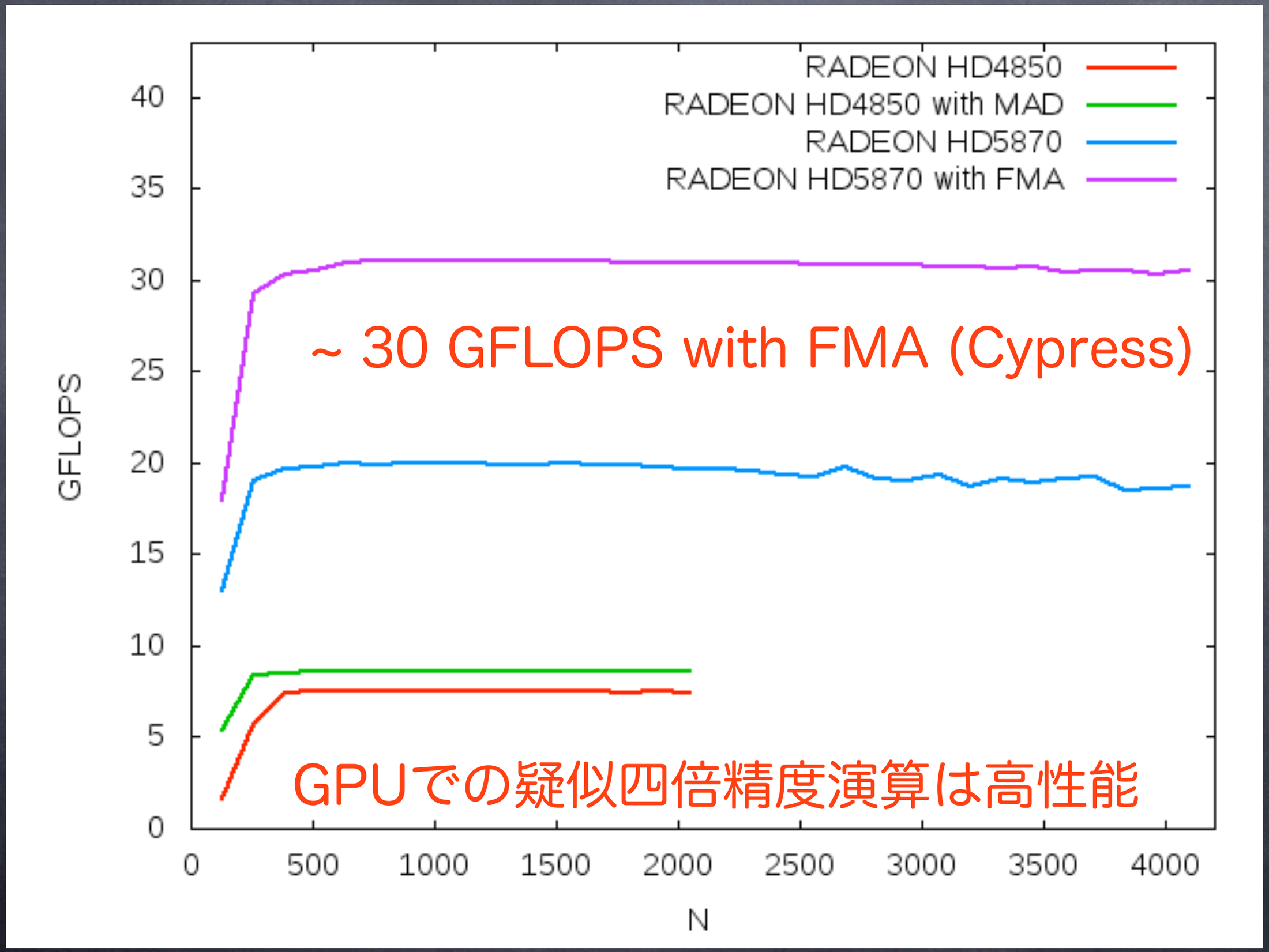
Matsumoto etal. 2011

大きい行列のGEMM



Matsumoto etal. 2011

四倍精度演算 行列積 (2010)



四倍精度演算 二重指数積分(2010)

演算性能　MFLOPS　~ 27 GFLOPS (Cypress)

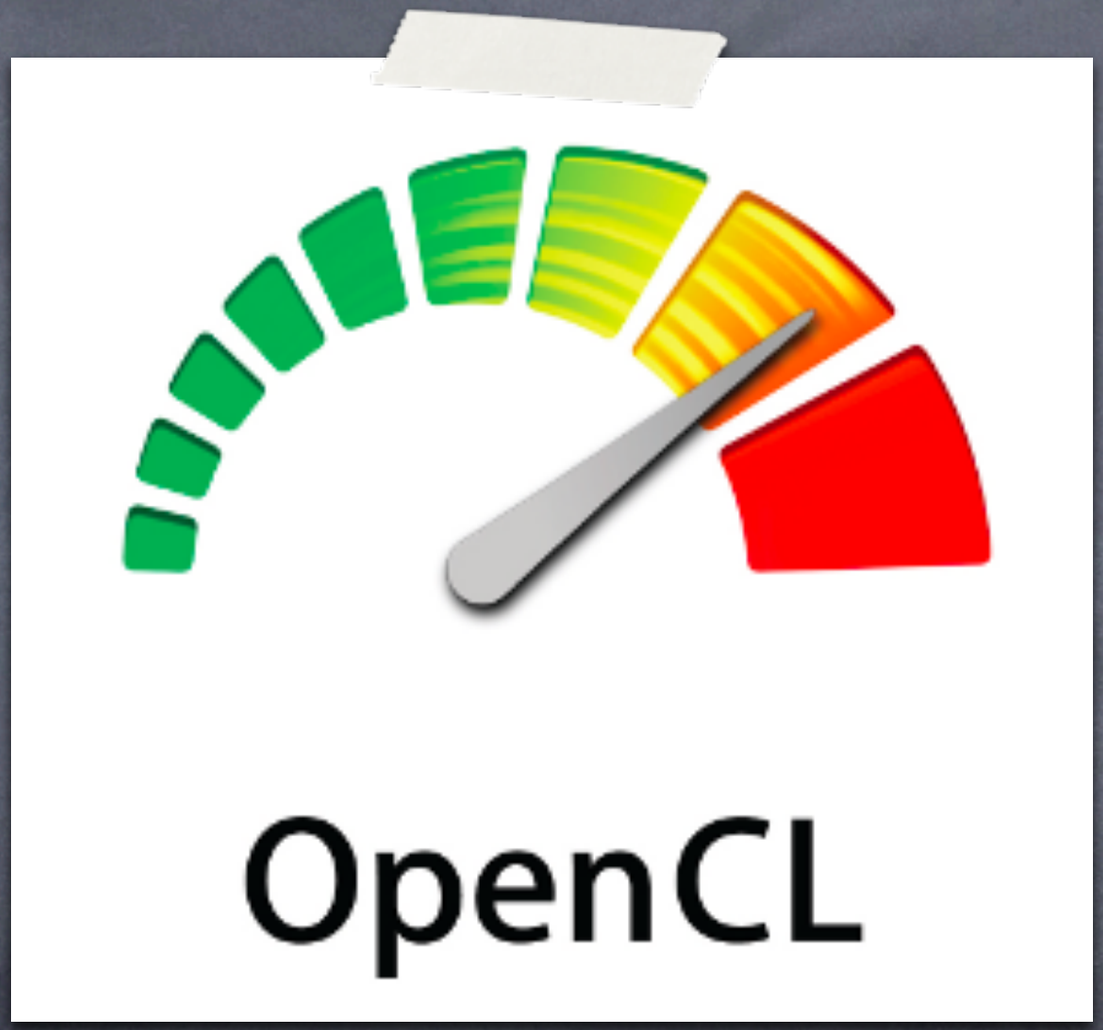
	N=256	N=512	N=1024	N=2048	note
Core i7	63.6	63.7	63.7		2670MHz 1 core
GRAPE-DR	2234	3106	3840	4365	380MHz 512 core
RV770	5220	5694	5977	6058	850MHz 160 core
Cypress	9395	12958	15497	16938	850MHz 320 core
Cypress FMA			23981	27270	850MHz 320 core

消費電力あたり

	MFLOPS	MFLOPS/W	W (nominal)
Core i7	63.7	2.12	30W
GRAPE-DR	4365	87.3	50W
RV770	6058	37.9	160W
Cypress FMA	27270	145.1	188W

AMD GPUでの成果のまとめ

- 我々のグループで様々な成果
  - 2010年まではILによるプログラミング
    - 低レベル(アセンブリ言語)だがほぼ最高性能
    - レジスタ割り当ての必要がないのでそれほど難しくはない
    - N-body, GEMM, 四倍精度の高速化
  - 2011年以降OpenCLが安定高速化
- GPUではShared memoryが必須?
  - というのは単なる思い違い
  - NVIDIAのGPUでもimage bufferを使う方が速い



OpenCLでの最新の研究成果

OpenCLについて

- 並列計算APIの標準規格
  - AMD, Apple, IBM, Intel, NVIDIA etc...
  - CPU, GPU, CellBE, DSP, FPGA etc...
- アクセラレータのプログラミングモデル
  - ホスト計算機から「デバイス」の機能を呼び出す
    - デバイスで動作するコードを「カーネル」と呼ぶ
    - CUDAと本質的な差はない
  - SIMD的な動作するスレッド群の並列化
  - 演算の明示的なベクトル化

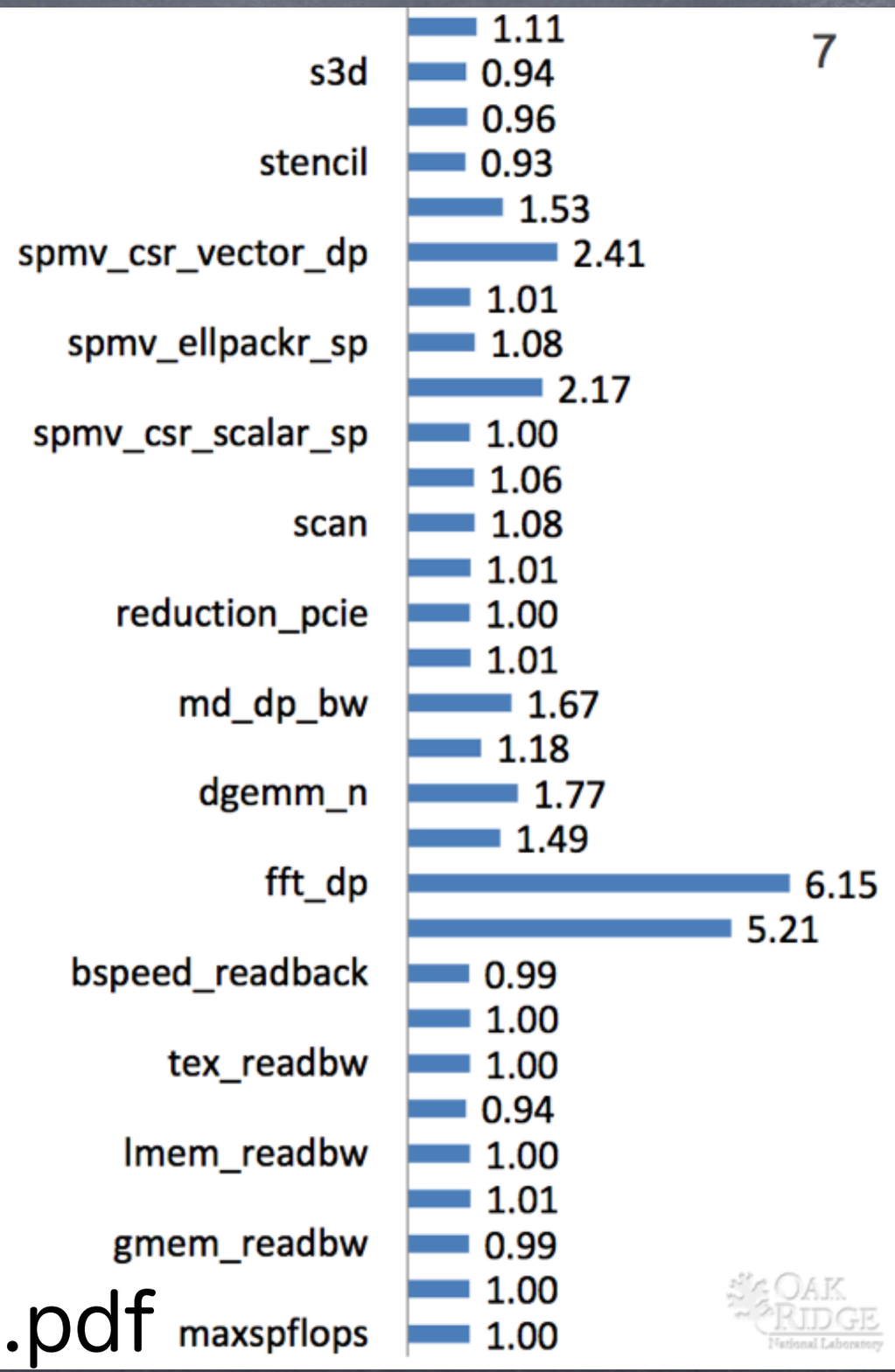
OpenCLの利点と欠点

- 利点
  - 各社のデバイスで動作する
  - CPUとGPUを同時扱うことが可能
    - 従来: SSEベクトル化 + OpenMP/pthread + MPI
    - OpenCL時代:
      - OpenCL(ベクトル変数 + スレッド) + MPI
      - (OpenCL+OpenMP) + MPI
- 欠点
  - より煩雑なプログラミング: C++ APIは簡単
  - NVIDIAのデバイスでは性能が落ちる場合あり

OpenCLとCUDAの性能比較

CUDA and OpenCL

- What does performance look like today?
- This chart shows the speedup of CUDA over OpenCL on a single Tesla M2070 on KIDS (CUDA 4.0)
- Note that performance is (in most cases, close to equivalent)
- Cases where it's not tend to be related to texture memory or transcendental intrinsics



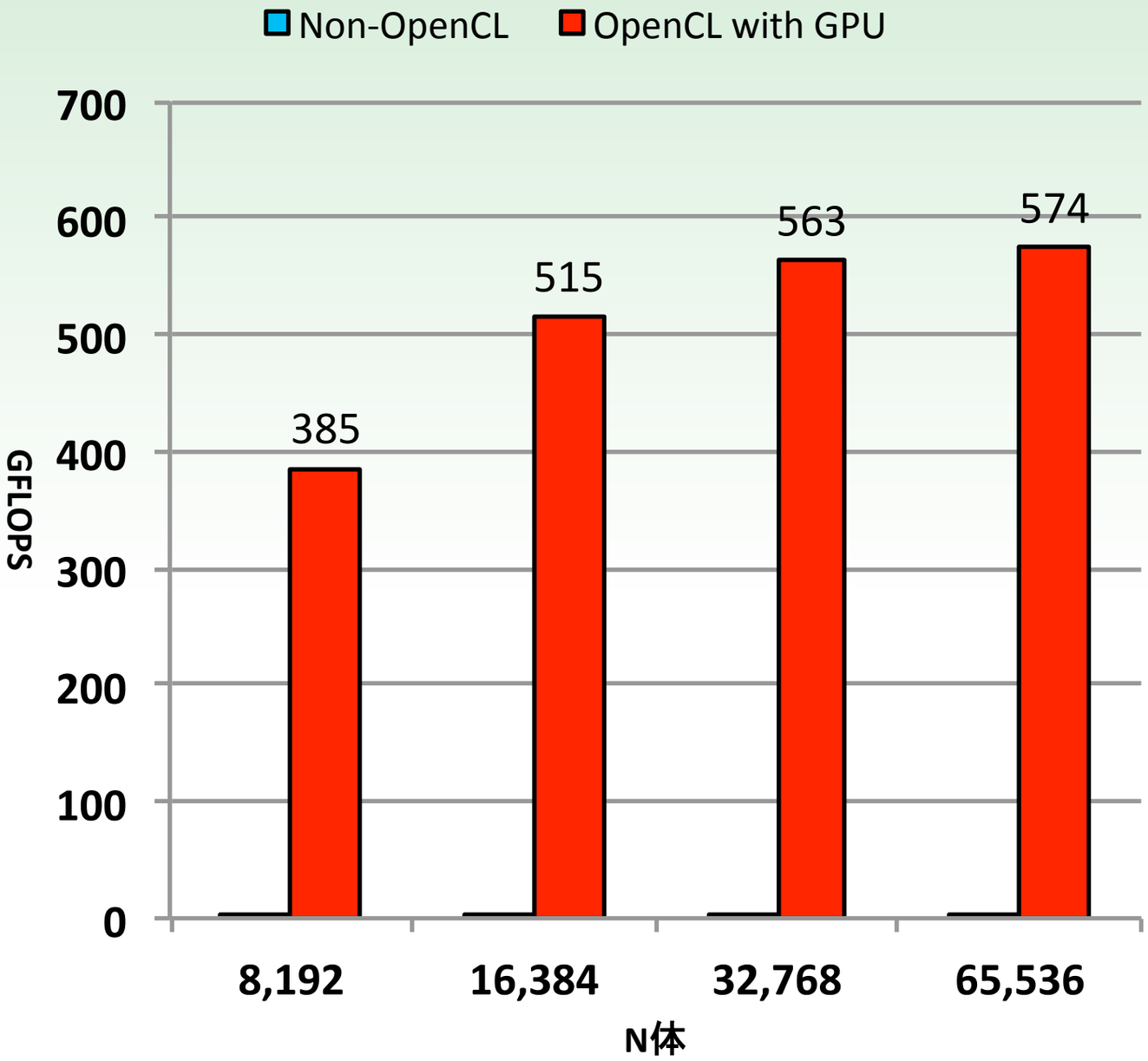
from Spafford 13-shoc.pdf

http://keeneland.gatech.edu/2012-02-20-workshop

OpenCLの評価:N-Body $O(N^2)$ (1)

2011年度卒業研究 鈴木Y

GPUとOpenCLの有効性



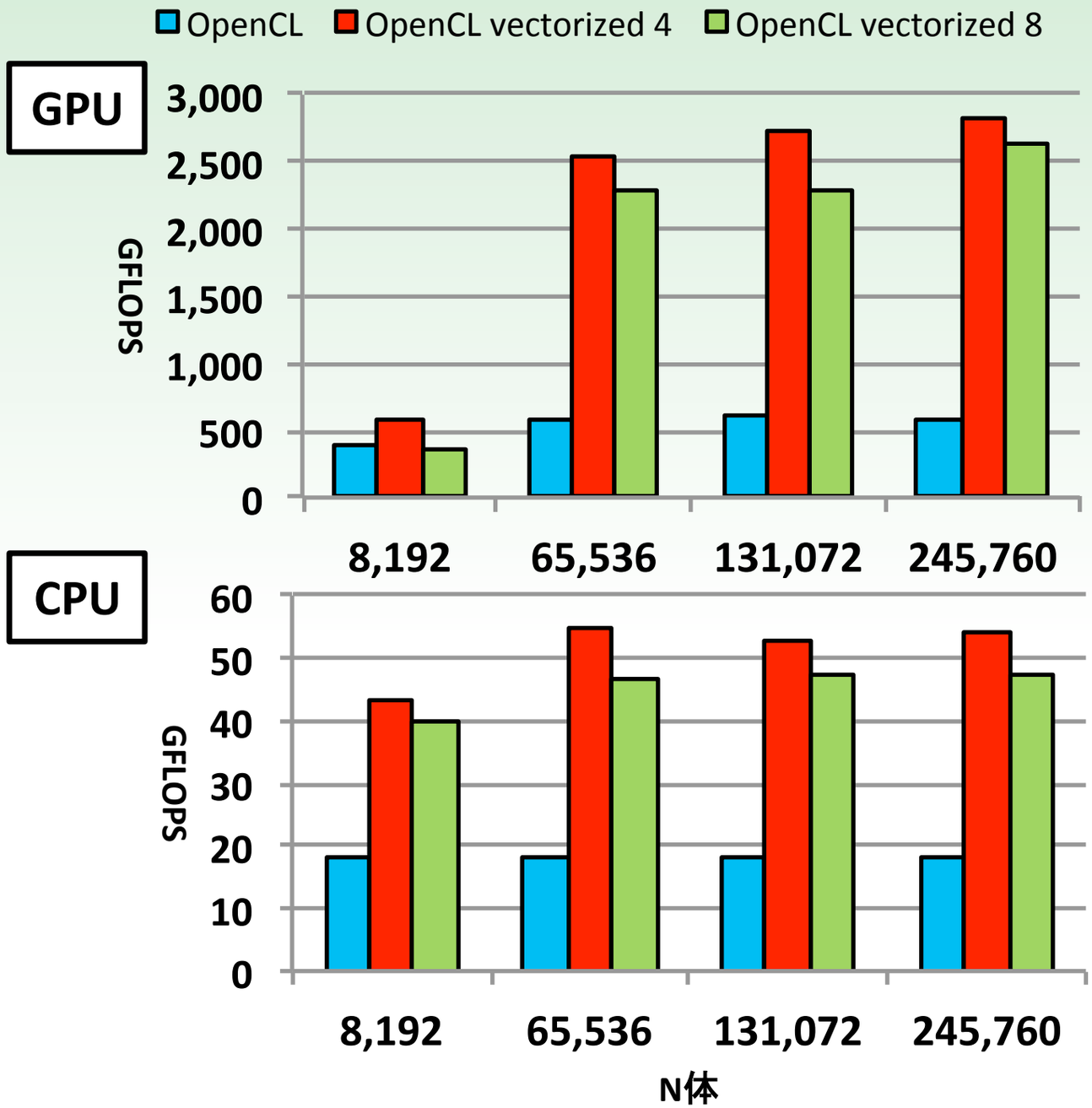
	System A
Maker	AMD
CPU	Phenom II X6 1090T
N_{core}	6
N_{thread}	6
clock	3.2 GHz
memory	4GB
SP(SSE)	153.6 GFLOPS
GPU	HD5870
SSE of GPU	2.72 TFLOPS

理論性能と出力性能の比較:

OpenCL with GPU → 0.21

OpenCLの評価:N-Body $O(N^2)$ (2)

GPUとCPUの性能比較・及びベクトル化の有効性



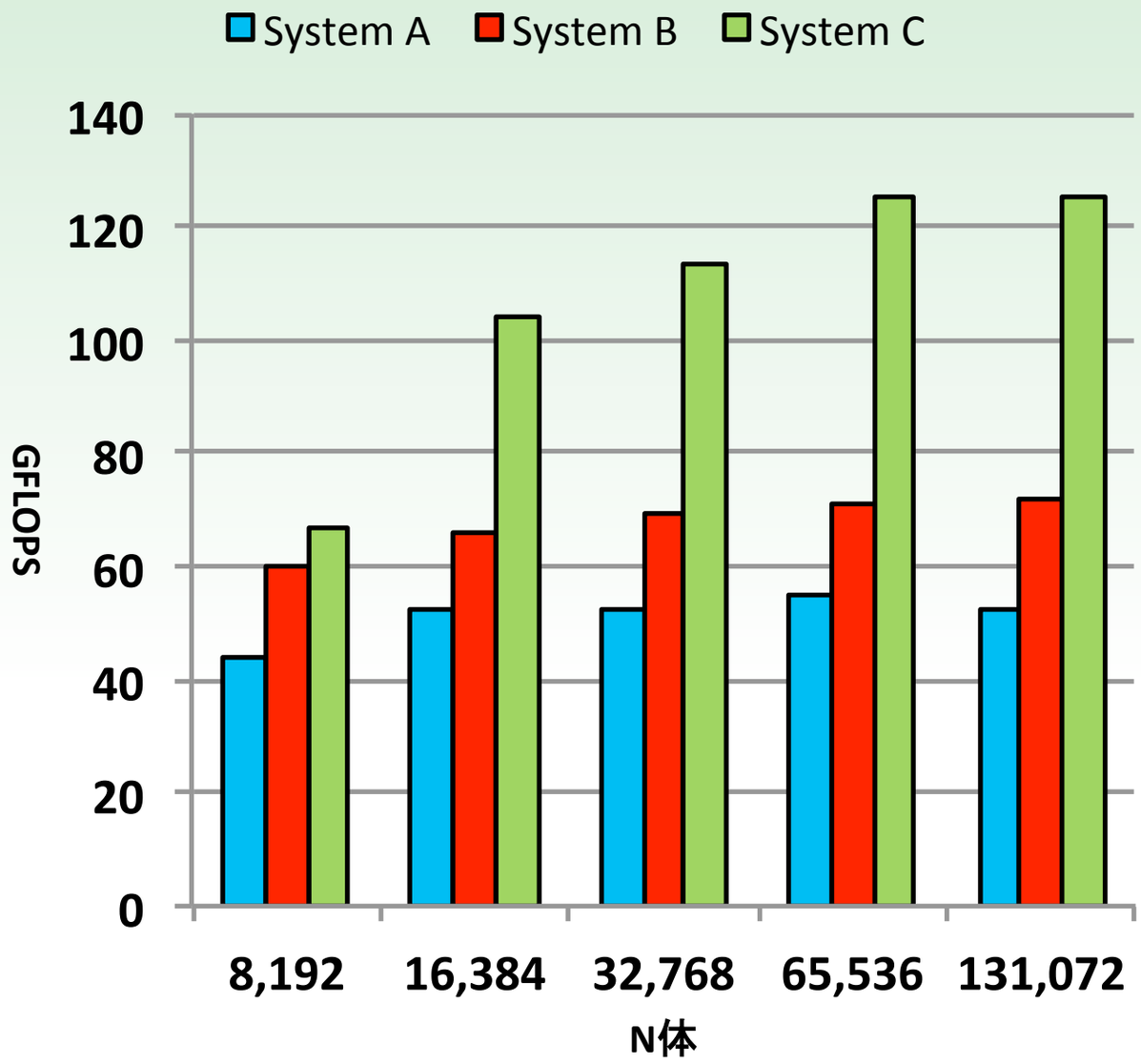
	System A
Maker	AMD
CPU	Phenom II X6 1090T
N_{core}	6
N_{thread}	6
clock	3.2 GHz
memory	4GB
SP(SSE)	153.6 GFLOPS
GPU	HD5870
SSE of GPU	2.72 TFLOPS

理論性能と出力性能の比較:

	OpenCL	OpenCL vectorized 4	OpenCL vectorized 8
GPU	0.21	0.99	0.97
CPU	0.12	0.36	0.31

OpenCLの評価:N-Body $O(N^2)$ (3)

3つの異なるマシンにおいての性能比較



	System A	System B	System C
Maker	AMD	Intel	AMD
CPU	Phenom II X6 1090T	Intel Core i7-2600K	Opteron Processor 6168 x 2
N_{core}	6	4	24
N_{thread}	6	8	24
clock	3.2 GHz	3.4 GHz	1.9 GHz
memory	4GB	16GB	32GB
SP(SSE)	153.6 GFLOPS	108.8/217.6 GFLOPS	364.8 GFLOPS

理論性能と出力性能の比較:

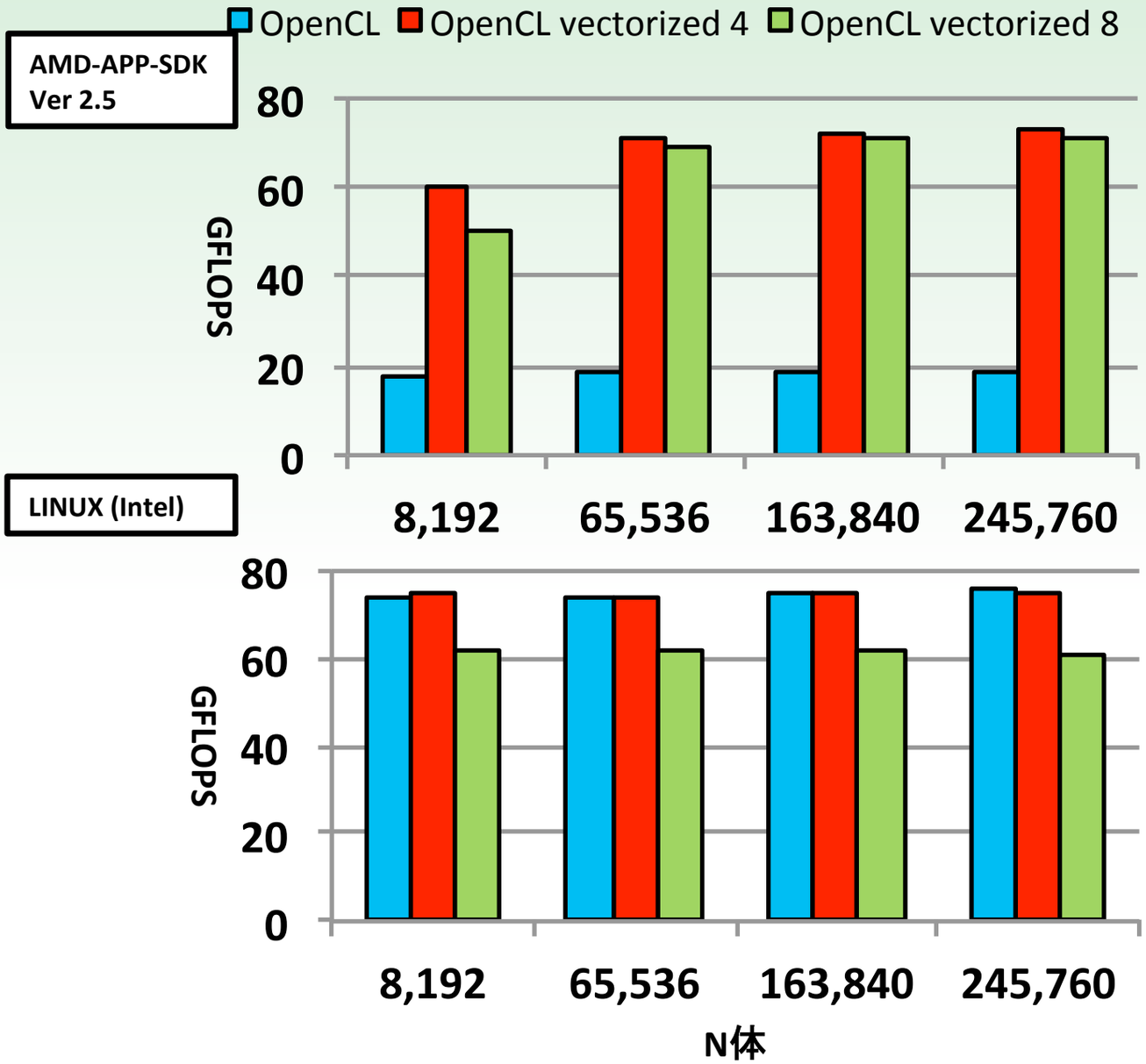
System A	System B	System C
0.36	0.34	0.35

CPUの性能(core数)に比例した性能

OpenCLの評価:N-Body $O(N^2)$ (4)

異なるSDKにおける性能比較

IntelのSDKは自動ベクトル化機構がある



	System B
Maker	Intel
CPU	Intel Core i7-2600K
N_{core}	4
N_{thread}	8
clock	3.4 GHz
memory	16GB
SP(SSE)	108.8/217.6 GFLOPS

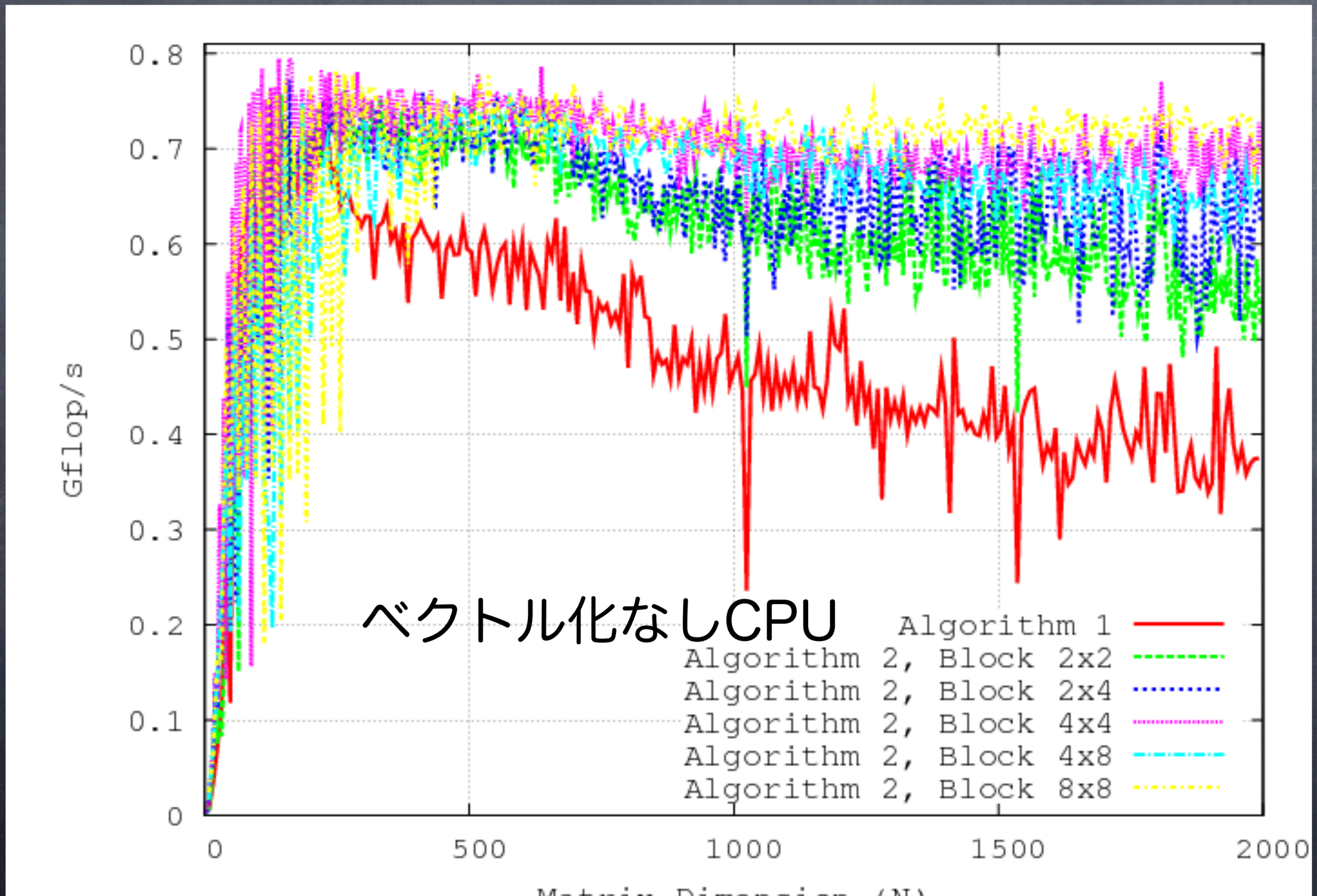
理論性能と出力性能の比較:

	OpenCL	OpenCL vectorized 4	OpenCL vectorized 8
AMD SDK	0.09	0.33	0.33
Intel SDK	0.35	0.34	0.28

OpenCLの評価:四倍精度演算(1)

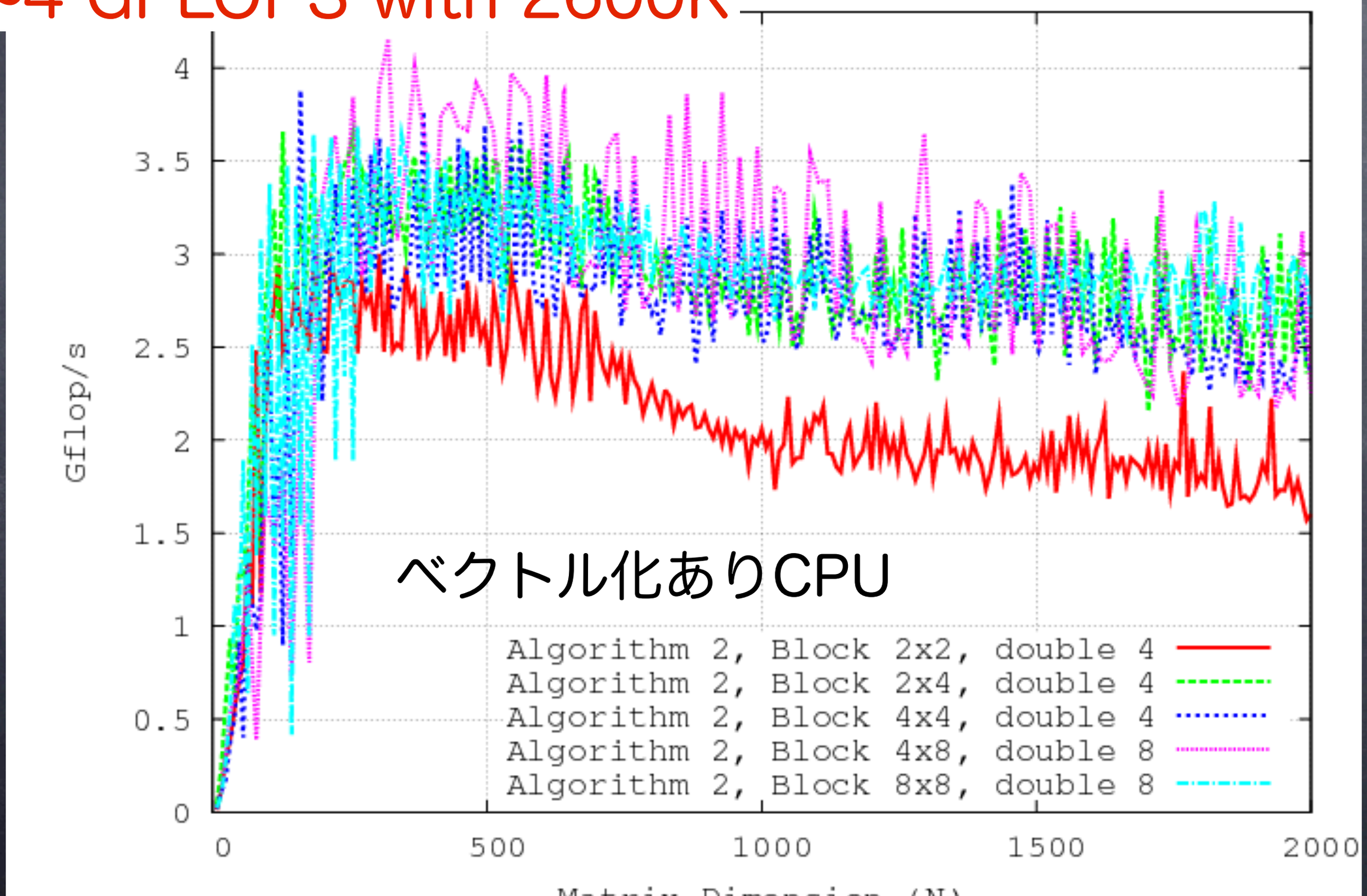
- 2011年度卒業研究 中村
- QDライブラリをOpenCLで実装
  - 現時点では四倍精度のみ
- 行列演算に応用：行列積とLU分解
- 主な最適化手法
  - ブロッキング
  - ベクトル化
  - FMAの利用
- 3月のHPC研究会(有馬温泉)で発表

OpenCLの評価:四倍精度演算(2)



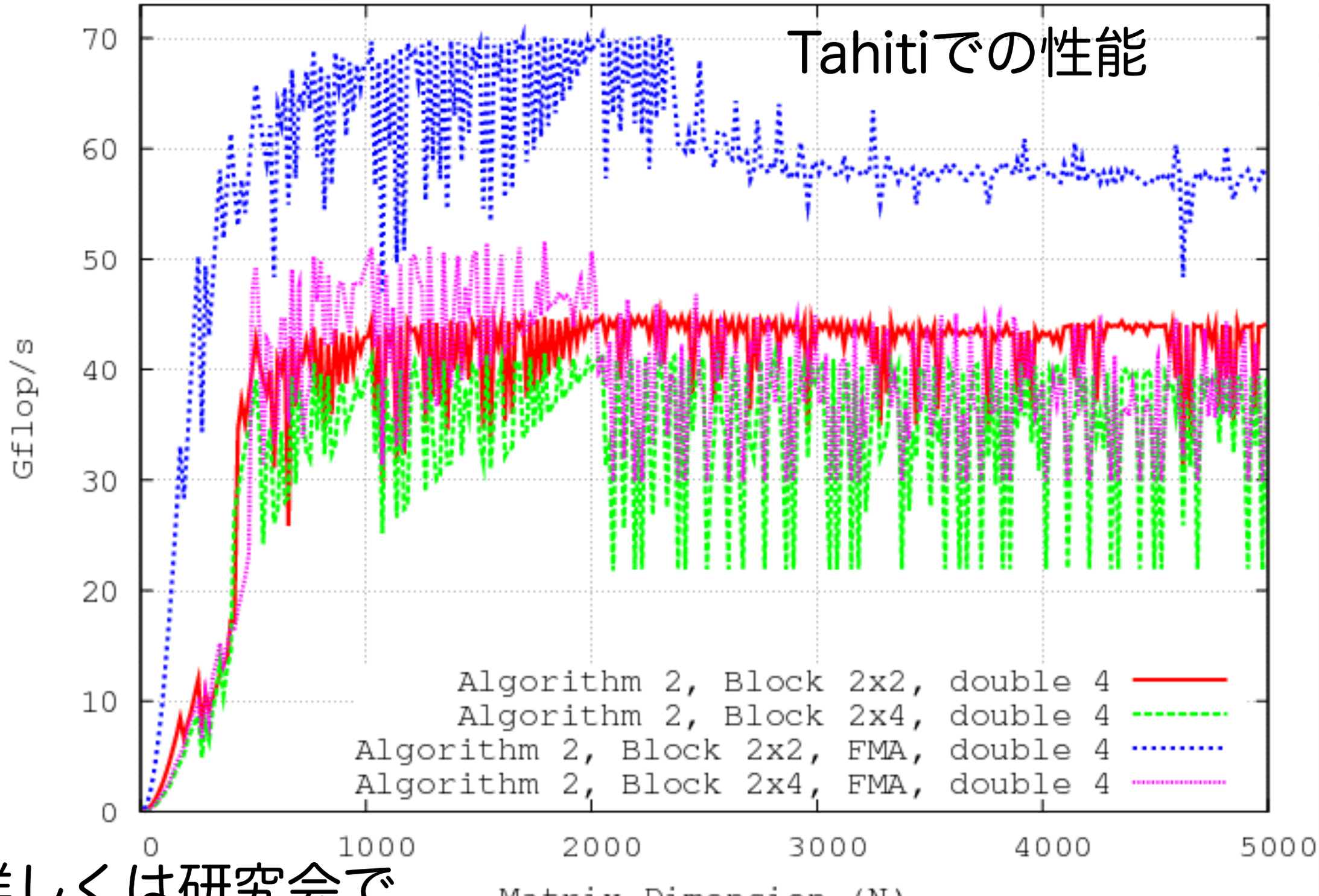
OpenCLの評価:四倍精度演算(3)

~4 GFLOPS with 2600K



OpenCLの評価:四倍精度演算(4)

~70 GFLOPS



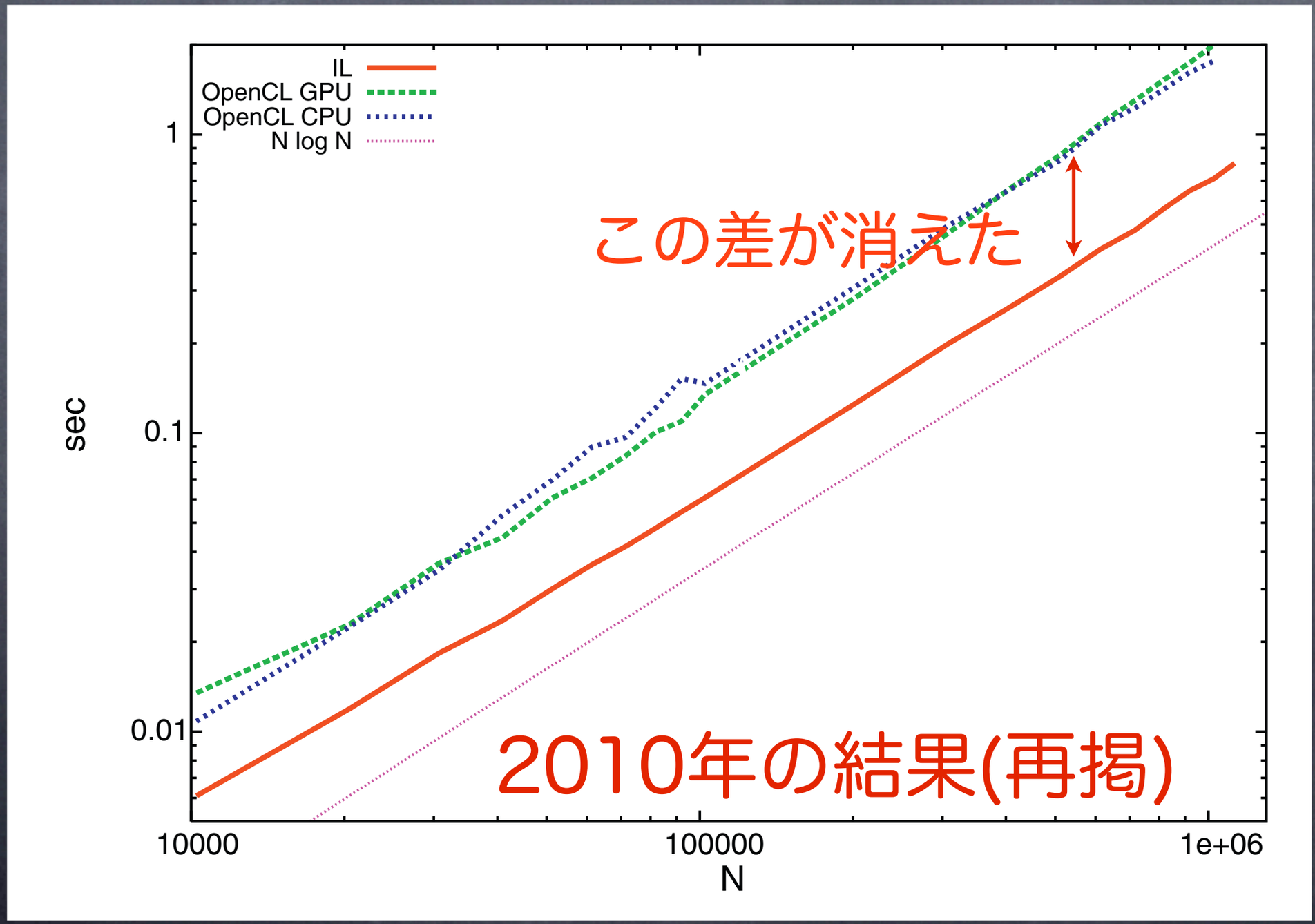
詳しくは研究会で

OpenCLの評価:文字列検索

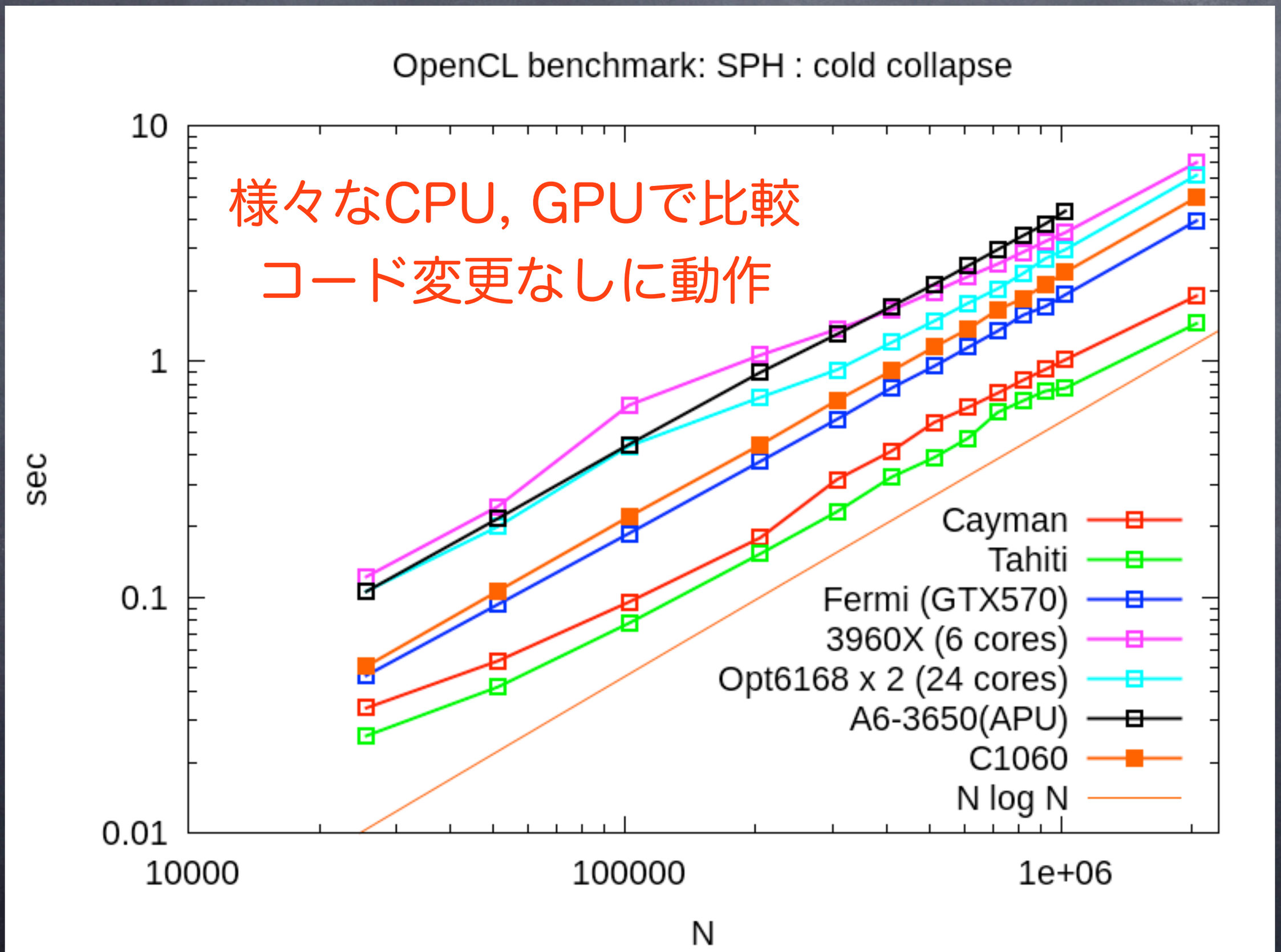
- 2011年度卒業研究 鈴木T (準備中)
- テキストの中から複数の文字列を並列で検索するカーネルをOpenCLで実装
  - いくつかの検索アルゴリズム
  - ベクトル化による高速化
  - grepとの性能比較
- 数値計算ではないアルゴリズムも高速化の余地は色々とある
  - 並列性や高帯域メモリの恩恵により

OpenCLの評価:Octree法 (1)

- 2011年のAMD OpenCL SDKでキャッシュの扱いが最適化されて高速化した



OpenCLの評価:Octree法応用 (2)



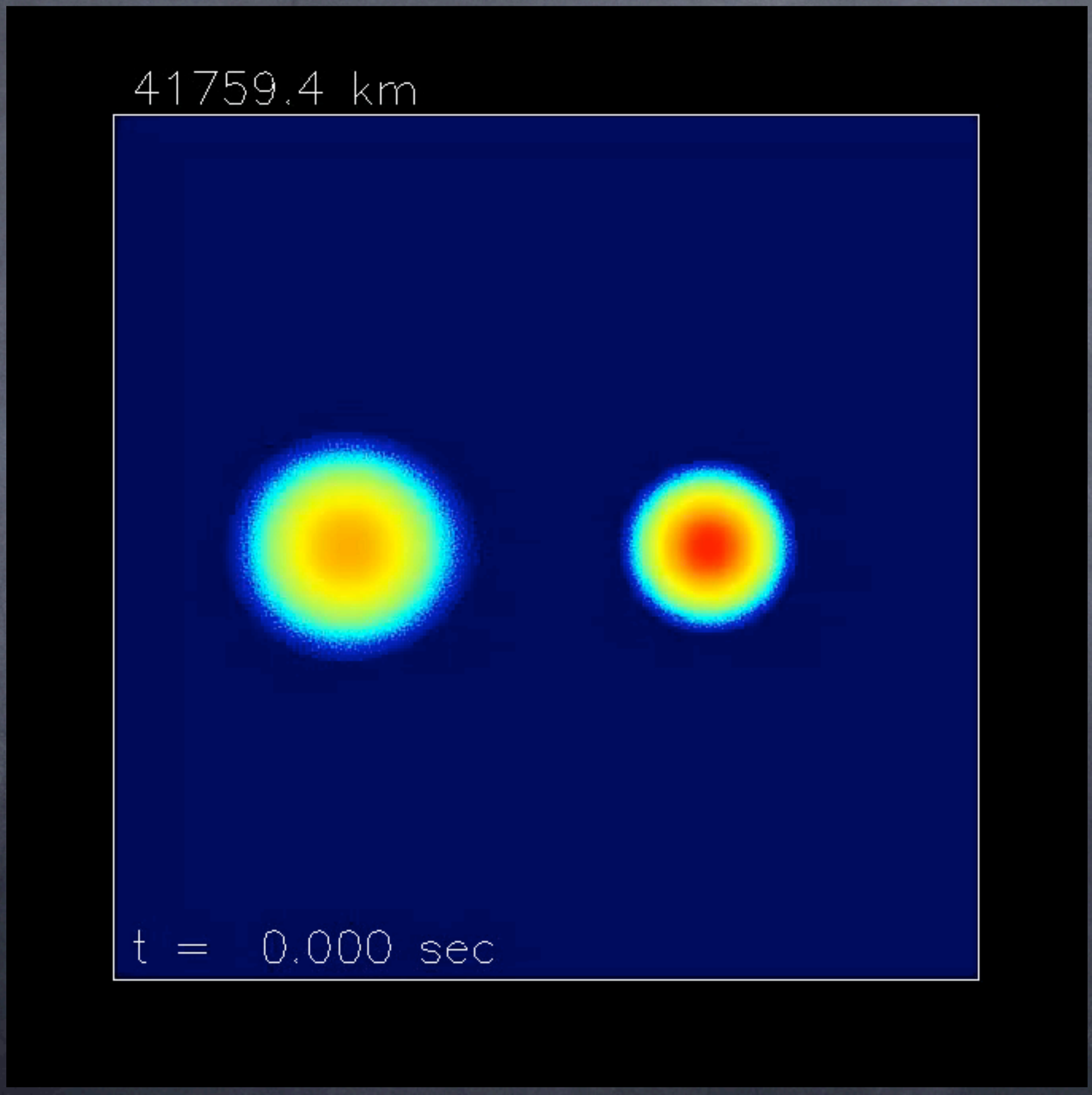
GPUでの最適化Octree

- Octreeで必要なステップ
  - tree構築 (並列化難)
  - treeノードの多重極モーメントの計算 (並列化難)
  - tree traversalによる相互作用計算 (並列化易)
- どの部分をGPUで実行するか？
  - 全てをGPUで実行する研究結果もある：遅い
    - O(N)の部分をGPUでやるのがよいのか問題
- 複数ノードでの並列化戦略

OpenCLの成果まとめ

- AMD GPUでのOpenCLは非常に有効
  - 2010年度まではILの利用が必須だった
- OpenCLによるCPUプログラミング
  - 効果的に利用可能
  - 特にSSE/AVXが容易にプログラミングできる
  - Intel SDKの自動ベクトル化は効果的
- NVIDIA GPUでのOpenCL実装
  - 現時点ではCUDAのほうがより最適化されている

白色矮星の衝突計算



メタプログラミングとHPC

高速計算とメタプログラミング

- 伝統的な高速計算(HPC) = Fortran
  - この言語自体が数値計算のために設計された
  - 最適化がやりやすい言語仕様
- 現代の高速計算 = 並列計算
  - プログラミングが本質的に困難
  - 自動並列化は絶望的
    - Fortranのベクトル化だけはうまくいっていたけれど。。。
  - 上から下まで異なる粒度の並列化
    - SSE/AVX, スレッド, CPU-GPU, MPI....
  - 記述量を減らす意味でメタプログラミング重要

Domain Specific Language

- これもメタプログラミング
  - HPCの世界でも近年色々やられはじめている
- LSUMP (LLVM-SUMP)
  - Nakasato & Makino (2009)
  - 粒子法シミュレーションに特化したDSL
  - 元々GRAPE-DR用のコンパイラとして設計
    - GRAPE-DRは粒子シミュレーションに向いた準汎用計算機
  - 後にGPUにも対応。
  - 単、倍、四倍精度演算のカーネルを生成可能

メタプログラミングと言語

- メタプログラミングによる記述量の削減は、並列計算必須のHPCにますます欠かせなくなる
  - 予想：Fortranのような言語は徐々に廃れる
  - C++の重要性が増す：OOPではなくGPの面で
  - 例1 Eigen (C++行列配列ライブラリ)
    - Template meta programmingでSSE/AVX対応：MKL等と同性能
  - 例2 PETSc (C++ HPC用並列ライブラリ)
    - Template meta programmingで並列処理に対応
    - 演算子オーバーロードを利用したターゲットごとのコード生成

モダンな言語とHPC

- そのものでHPC計算はまだ無理
- DSLの実装言語としては有力
  - 例1：Liszt (偏微分方程式のためのDSL)
    - Scalaで実装されている
    - DSLからCUDA, MPIを含むC++のコードを生成
  - 例2：Paraiso (Euler的流体スキーム用DSL)
    - Haskellで実装されている
    - DSLからCUDA, OpneMPを含むC++のコードを生成
    - http://www.paraiso-lang.org/

自動チューニング

- DSLと対になるのが自動チューニング
  - DGEMMカーネルのRubyによる生成とチューニング

```ruby
if order == 'R' and transa == 'N' and transb == 'N' # RNN
  w << "__local #{vec_type} __a[#{k_width}][#{blk_m}];\n" if use_shared_a
  w << "__local #{vec_type} __b[#{blk_k}][#{blk_n/vec_width}];\n" if use_shared_b
  w << "const int i = get_global_id(1) << #{sblk_mshift};\n"
  w << "const int j = get_global_id(0) << #{shift_hash[sblk_n/vec_width]};\n"
  if use_shared_a
    w << "const int __i = get_local_id(1) << #{sblk_mshift};\n"
    w << "const int __j = get_local_id(0) << #{shift_hash[blk_k/(blk_n/sblk_n*vec_width)]};\n"
  elsif
    w << "const int __i = get_local_id(1) << #{shift_hash[blk_k/(blk_m/sblk_m)]};\n"
    w << "const int __j = get_local_id(0) << #{shift_hash[sblk_n/vec_width]};\n"
  end

  w << "#{vec_type} c[#{sblk_m}][#{sblk_nd}];\n"
  (0...sblk_m).each do |y|
    (0...sblk_nd).each do |x|
      w << "c[#{y}][#{x}] = (#{vec_type})0.0;\n"
    end
  end

  if use_shared_a or use_shared_b
    w << "for (int l = 0; l < k; l += #{blk_k}) {\n"

    if use_shared_a
      (0...sblk_m).each do |y|
        (0...[blk_k/(blk_n/sblk_n*vec_width),1].max).each do |x|
          w << "__a[__j+#{x}][__i+#{y}] = A[(i+#{y})*(lda>>#{vwshift})+((l>>#{vwshift})+__j+#{x})];\n"
        end
      end
    end
    if use_shared_b
      (0...[blk_k/(blk_m/sblk_m),1].max).each do |y|
        (0...sblk_n/vec_width).each do |x|
          w << "__b[__i+#{y}][__j+#{x}] = B[(l+__i+#{y})*(ldb>>#{vwshift})+j+#{x}];\n"
        end
      end
    end
    w << "barrier(CLK_LOCAL_MEM_FENCE);\n"

    w << "for (int p = 0; p < #{blk_k}; p += #{sblk_k}) {\n"

    w << "#{vec_type} a[#{sblk_m}], b[#{sblk_nd}];\n"
    (0...sblk_k).each do |z|
      if z%vec_width == 0
```

gen_dgemm_kernel.rb

松本さん (会津大学大学院)

まとめ

- CPU/GPU混在システムの評価
- AMDのGPUでの性能評価
  - 「ソフトウエアキャッシュ」なしで高性能
- OpenCLでの様々な問題の性能評価
  - GPUもCPUも有効活用可能
  - OpenMPに変わるノード内並列化手法になり得る
  - SSE/AVXの有効利用, 自動ベクトル化
- HPCではメタプログラミングが重要に