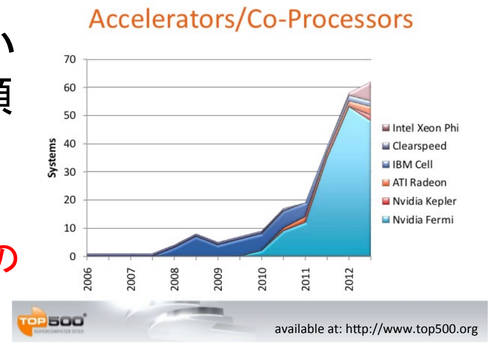
jh130042-IS02

松岡 聡 (東京工業大学)

OpenACCを用いた大規模流体アプリケーションの高速化

研究背景・目的

- メニーコアアクセラレータを用いたヘテロジニアスシステムの台頭
  - NVIDIA GPU, Intel mic など
  - ex. Titan, Tianhe, TSUBAME2.0
  - → CPU向けに作られた既存アプリの移植が課題



TITAN (2012) 27.1 Pflops
299,088 CPU cores
18,688 NVIDIA K20 GPUs

TSUBAME (2010) 2.28 Pflops
17,984 CPU cores
4,224 NVIDIA Fermi GPUs

Stampede (2012) 3.96 Pflops
204,800+ CPU cores
6,400+ Intel Xeon Phi

- メニーコアアクセラレータ向けのプログラミング言語
  - CUDA・OpenCL
    - 現在最も広く使われている
    - 低級な記述が必要
  - OpenACC
    - 指示文の挿入による
    - 多くのメニーコアに対応（予定）
    - GPU環境への移植の簡素化に期待

```
!$acc kernels present(a, b, c)
!$acc loop
  do j = 1, n
!$acc loop
    do i = 1, n
      cc = 0
      do k = 1, n
        cc = cc + a(i,k) * b(k,j)
      end do
      c(i,j) = cc
    end do
  end do
!$acc end kernels
```

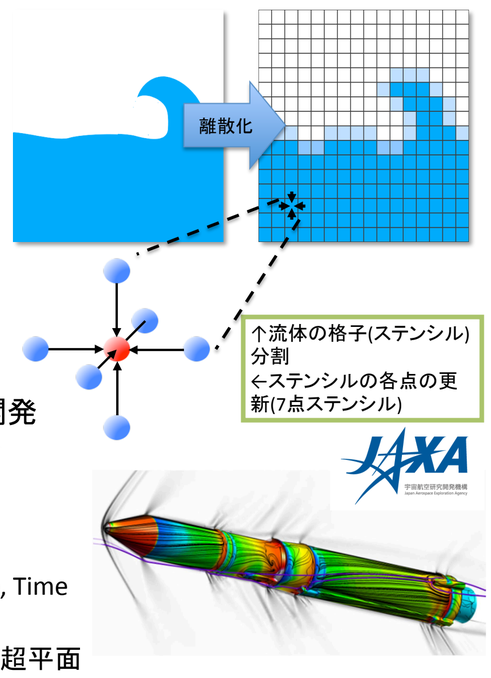
OpenACCによる行列積

- 研究目的
  - 数値流体アプリケーションのメニーコア環境への移植手法の確立
    - 既存のCPUアプリケーションのGPU環境への移植における問題点の抽出
    - OpenACCを用いた移植における問題点の抽出とその解決

評価手法

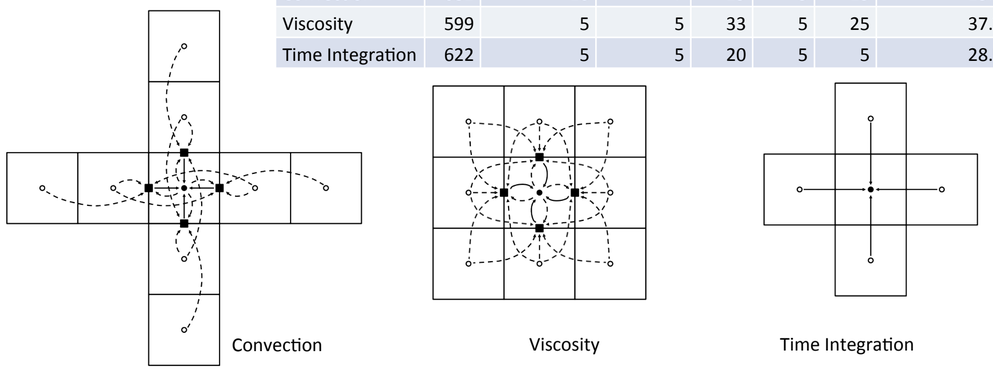
CUDA・OpenACCを用いてカーネルベンチマークと実アプリケーションの移植・最適化を行いCUDAと比較評価

- カーネルベンチマーク
  - 行列積
    - 計算バウンドなカーネル
  - 7点ステンシル
    - 流体の差分計算に利用
    - 単精度，メモリバウンドなカーネル
- 実アプリケーション
  - UPACS
    - 宇宙航空研究開発機構により研究・開発
    - 航空宇宙分野の流体解析ソフトウェア
    - 約10万行のFortran90からなる
    - 3つの主要フェーズからなる
      - Convection(陽解法), Viscosity(陽解法), Time Integration(陰解法)
      - Time Integration にはMFGS法を用い，超平面法により並列化



主要3フェーズ

Phase	LoC	# of subroutines	# of loop nests	# of 3-D data Read	Upload	Temporary	% of Total time (1CPU core)
Convection	682	10	7	29	5	15	25.0%
Viscosity	599	5	5	33	5	25	37.7%
Time Integration	622	5	5	20	5	5	28.5%



実験・性能評価

- 実験環境
  - TSUBAME2.0 Thinノードを利用
  - 3社のコンパイラを利用
- 性能評価実験
  - それぞれのアプリケーションを移植・最適化
  - OpenACCの3コンパイラとCUDAの性能比較
  - グラフ・表はそれぞれ性能・変更行数

実験環境(TSUBAME2.0 Thinノード)

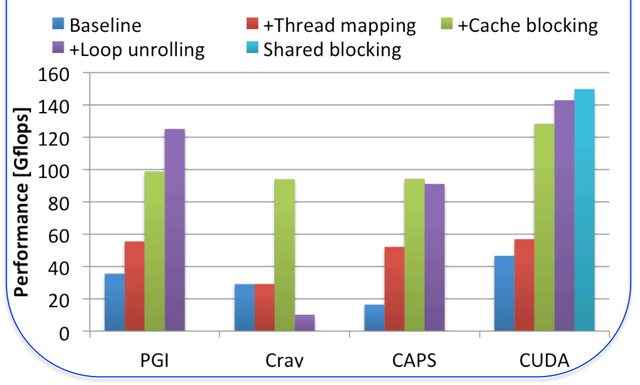
	CPU	GPU
Type	Intel Xeon Westmere-EP 2.9GHz	NVIDIA Fermi M2050
Cores	6 core	14SM, 448 CUDA core

使用コンパイラ

	compiler version	option
PGI	pgfortran 12.10-0	OpenACC: -O3 -acc -ta=nvidia,cc20,keepgpu CUDA Fortran: -O3 –Mcuda=cc20
Cray	ftn 8.1.0.165	-O3 -h accel=nvidia_20 -h system_alloc
CAPS	hmpp 3.3.0 host compiler ifort 11.1	--nvcc-options -arch,sm_20 --nvcc-options -O3 ifort -O3
CUDA C	nvcc 4.1	-O3 –arch=sm_20
Fortran	ifort 11.1	-O3 -static -xP -openmp

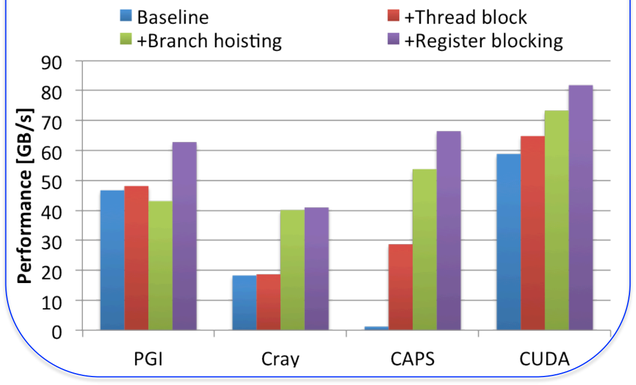
行列積

	Baseline	Thread mapping	Cache blocking	Loop unrolling	Shared blocking
OpenACC	9	11	62	302	
CUDA	26	26	77	317	45



7点ステンシル

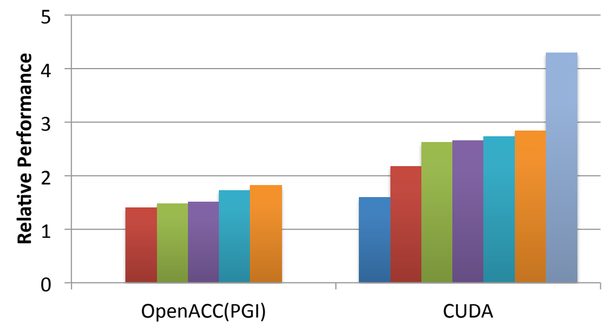
	Baseline	Thread mapping	Branch Hoisting	Register blocking
OpenACC	7	10	18	29
CUDA	35	35	45	56

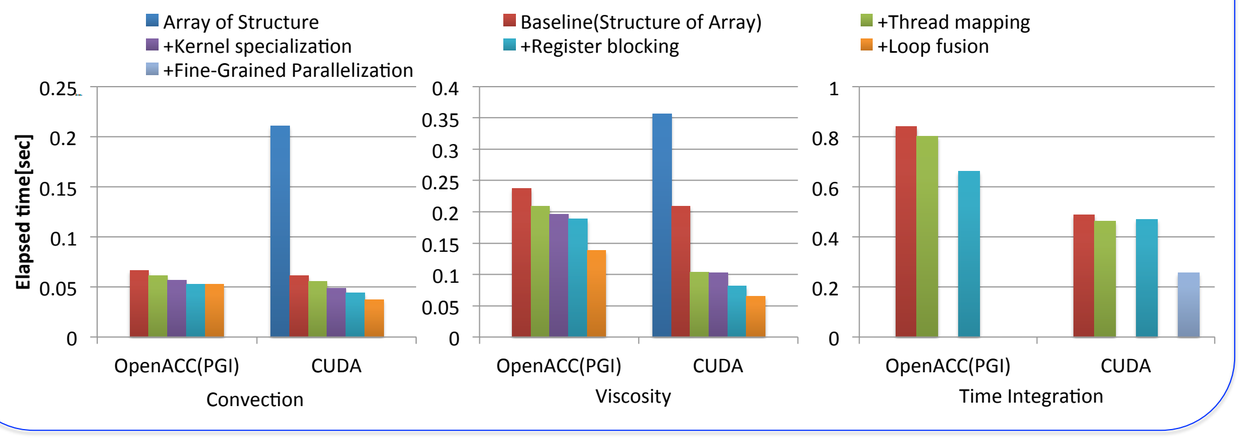


UPACS

- Baseline: Array of Structure
→ Structure of Array
- 問題サイズ：120^3
- グラフ
  - 右) OpenMPの6並列と比較した相対性能
  - 下)各フェーズの実行時間

	Baseline	Thread mapping	Kernel specialization	Register blocking	Loop fusion	MFGS
OpenACC	1788	1809	2601	2940	3296	
CUDA	5607	5743	6614	7641	8656	8870





- 性能評価
  - カーネルベンチマーク
    - PGIコンパイラでは，10行以下の書き換えで7~8割程度の性能
    - CUDAにおいて意味のある最適化は，OpenACCにおいても意味がある
  - 実アプリケーション
    - シェアードメモリを使った最適化で性能ギャップが大きくなる
    - OpenACCのベースラインにおいても，1800行程度書き換えている
    - Array of Structure から Structure of Array への変更の効果が大きく，OpenACCでは変更しなければ実行できない
  - まとめ
    - CPUとGPUのアーキテクチャの違いから、得意なデータ構造が異なるが、データ構造の書き換えがコストを大きくしている
- 今後の課題
  - データ構造を自動最適化する方法の提案を目指す

参考文献：[1]Tetsuya Hoshino, Naoya Maruyama, Satoshi Matsuoka, Ryoji Takaki, “CUDA vs OpenACC: Performance Case Studies with Kernel Benchmarks and a Memory-Bound CFD Application”, In Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2013), pp. 136-143, Delft, the Netherlands, May, 2013