Overview: “An Auto-Tuning Framework for Parallel Multicore Stencil Computations”

Shoaib Kamil et al., 2010

Presented by Matt Ziemann for UMD CMSC 714

03/23/2021

The Bottom Line

- **Generalized** stencil auto-tuning framework
  - Portable across varied architectures

- ~1.5-4x speedup vs conventional parallelization
  - Up to 22x speedup vs serial implementation

Auto-Tuning


Serial Code
Auto-Tuner
“Optimized” Parallel Implementation

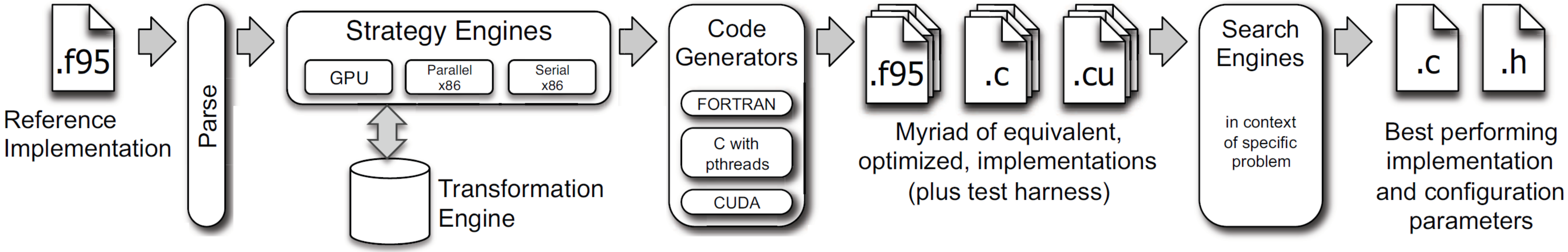
Generalized Auto-Tuning

Stencil Computations

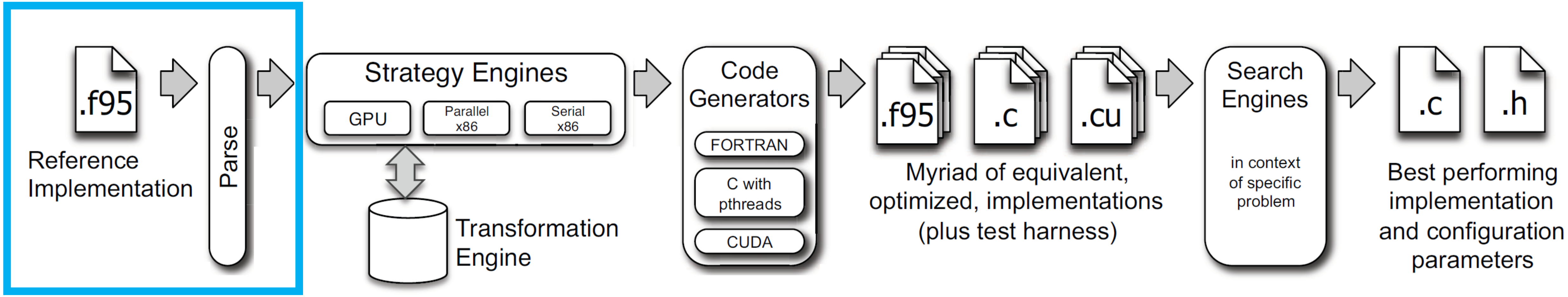
- Scientific computing applications
- Regular grid, nearest-neighbor computations
  - e.g. gradient, divergence, and Laplacian calculations
- High memory traffic for relatively low computation

- Previously, auto-tuning was successful only for single stencil instantiations, not multiple types of stencil computations
  - Required “immense effort” to hand-write

Stencil Auto-Tuning Framework

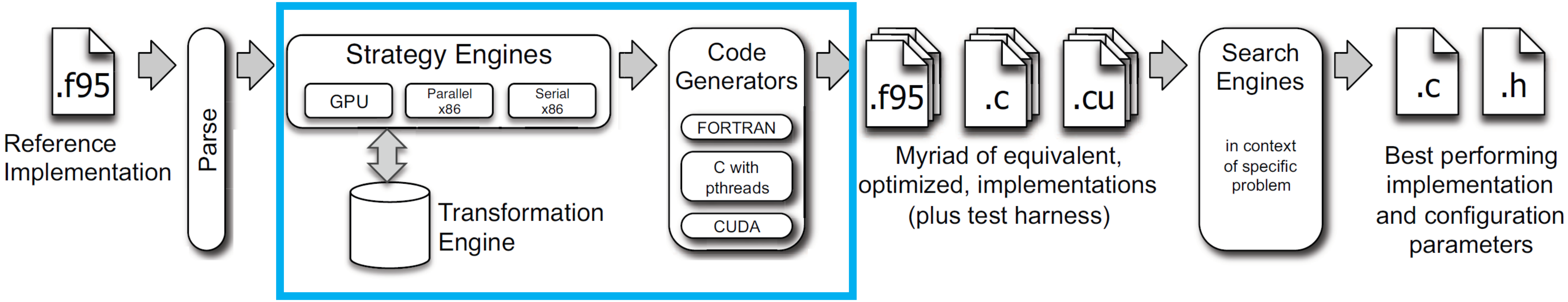


Stencil Auto-Tuning Framework



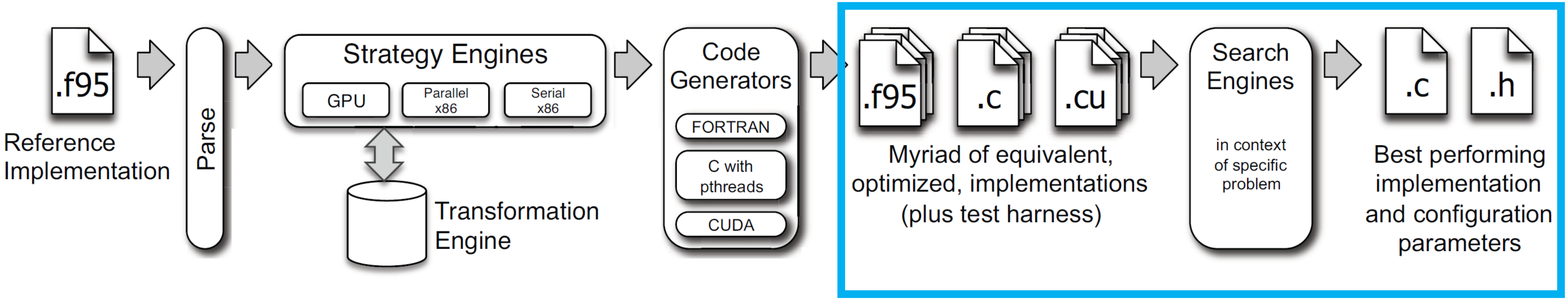
- Parses serial code and generates an *Abstract Syntax Tree* for later transformations
- Modular – different front-end implementations are possible (vs F95)

Stencil Auto-Tuning Framework



- Strategy engine intelligently searches parameter space of possible auto-tuned optimizations based on desired platform
- Transformation engine generates these variations
  - Uses domain specific-knowledge to implement safe optimizations that a conventional compiler cannot

Stencil Auto-Tuning Framework



- Search engine evaluates best-performing variation and passes that information to the user

Optimization Space

NX, NY, NZ = 256

Category	Optimization Parameter	Name	Barcelona/Nehalem	Victoria Falls	GTX280
			Parameter Tuning Range by Architecture		
Data Allocation	NUMA Aware		✓	✓	N/A
Domain Decomposition	Core Block Size	*CX*	NX	{8...NX}	$\{16^{\dagger}..NX\}$
		CY	{8...NY}	{8...NY}	$\{16^{\dagger}..NY\}$
		CZ	{128...NZ}	{128...NZ}	$\{16^{\dagger}..NZ\}$
	Thread Block Size	*TX*	CX	CX	$\{1..\frac{CX}{16}\}^{\ddagger}$
		TY	CY	CY	$\{\frac{CY}{16}..CY\}^{\ddagger}$
		TZ	CZ	CZ	$\{\frac{CZ}{16}..CZ\}^{\ddagger}$
	Chunk Size		$\{1 \ldots \frac{NX \times NY \times NZ}{CX \times CY \times CZ \times NThreads}\}$		N/A
Low Level	Array Indexing		✓	✓	✓
	Register Block Size	*RX*	{1...8}	{1...8}	1
		RY	{1...2}	{1...2}	1*
		RZ	{1...2}	{1...2}	1*

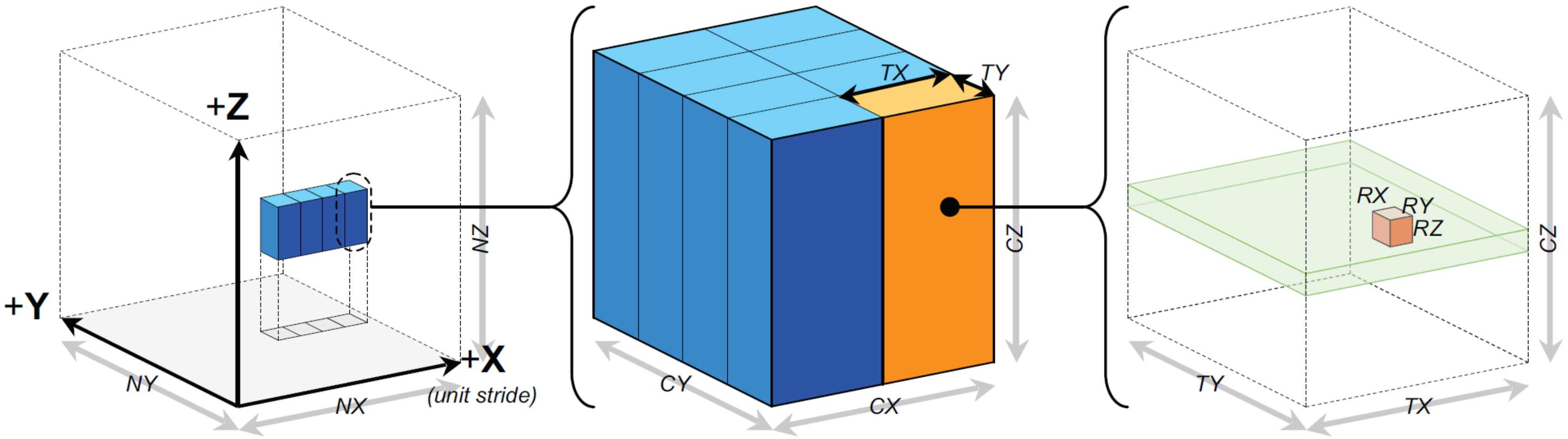
Optimization Space

NX, NY, NZ = 256

Category	Optimization Parameter	Name	Parameter Tuning Range by Architecture: Barcelona/Nehalem	Victoria Falls	GTX280
Data Allocation	NUMA Aware		✓	✓	N/A
Domain Decomposition	Core Block Size	CX	NX	{8...NX}	$\{16^{\dagger}..NX\}$
		CY	{8...NY}	{8...NY}	$\{16^{\dagger}..NY\}$
		CZ	{128...NZ}	{128...NZ}	$\{16^{\dagger}..NZ\}$
	Thread Block Size	TX	CX	CX	$\{1..\frac{CX}{16}\}^{\ddagger}$
		TY	CY	CY	$\{\frac{CY}{16}..CY\}^{\ddagger}$
		TZ	CZ	CZ	$\{\frac{CZ}{16}..CZ\}^{\ddagger}$
	Chunk Size		$\{1\ldots\frac{NX\times NY\times NZ}{CX\times CY\times CZ\times NThreads}\}$		N/A
Low Level	Array Indexing		✓	✓	✓
	Register Block Size	RX	{1...8}	{1...8}	1
		RY	{1...2}	{1...2}	1*
		RZ	{1...2}	{1...2}	1*

Domain Decomposition

NX, NY, NZ = 256



(a) Decomposition of a Node Block into a Chunk of Core Blocks

(b) Decomposition into Thread Blocks

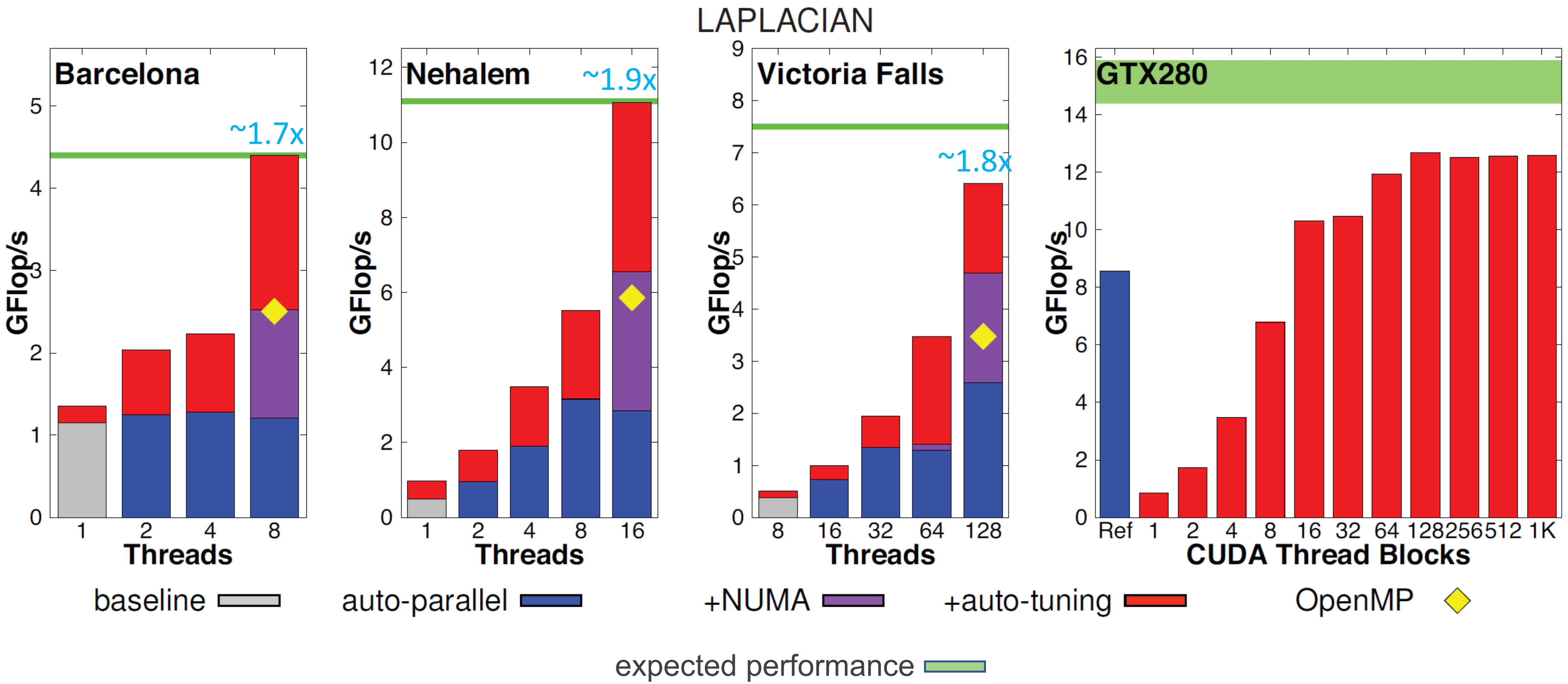
(c) Decomposition into Register Blocks

Category	Optimization Parameter	Name	Parameter Tuning Range by Architecture: Barcelona/Nehalem	Victoria Falls	GTX280
Domain Decomposition	Core Block Size	*CX*	NX	{8...NX}	$\{16^{\dagger}..NX\}$
		CY	{8...NY}	{8...NY}	$\{16^{\dagger}..NY\}$
		CZ	{128...NZ}	{128...NZ}	$\{16^{\dagger}..NZ\}$
	Thread Block Size	*TX*	CX	CX	$\{1..\frac{CX}{16}\}^{\ddagger}$
		TY	CY	CY	$\{\frac{CY}{16}..CY\}^{\ddagger}$
		TZ	CZ	CZ	$\{\frac{CZ}{16}..CZ\}^{\ddagger}$
	Chunk Size		$\{1\ldots\frac{NX\times NY\times NZ}{CX\times CY\times CZ\times NThreads}\}$		N/A

Performance Evaluation

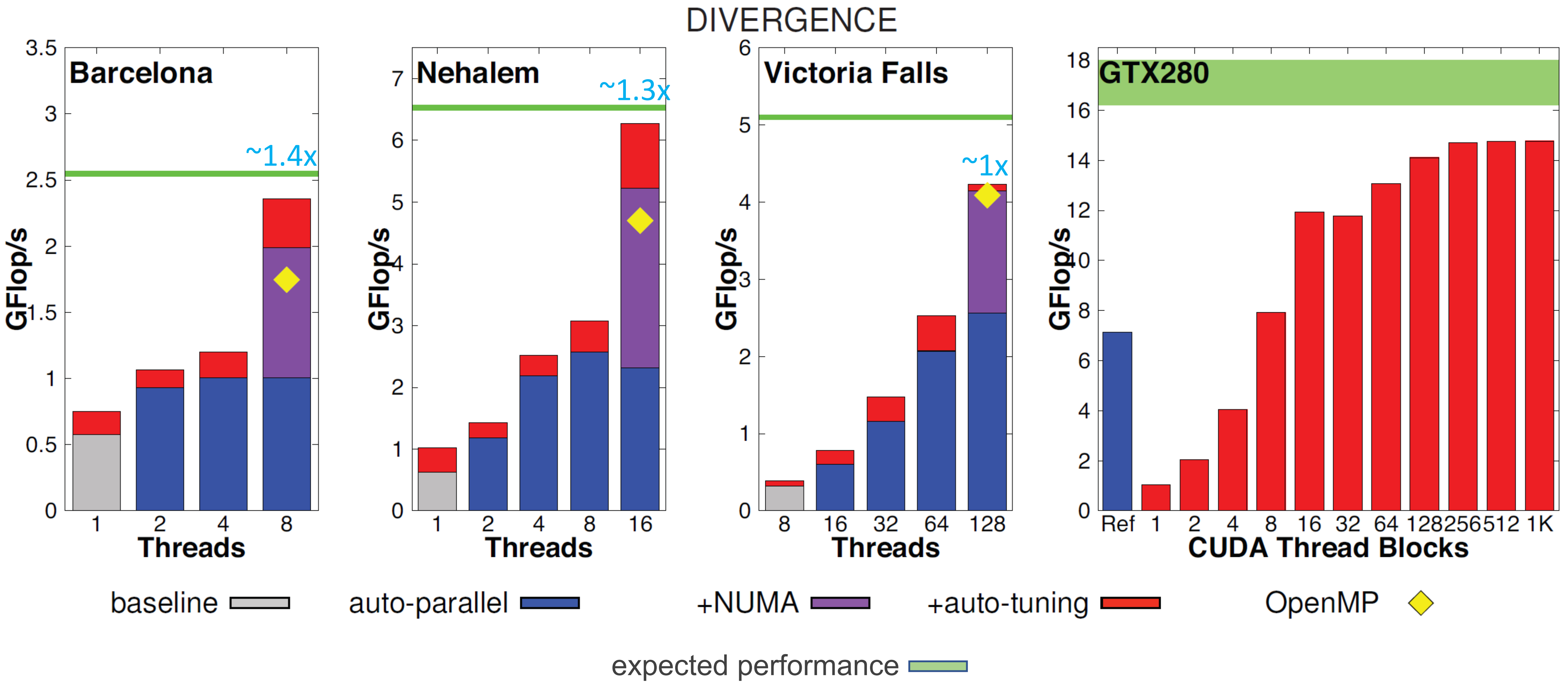
- Three stencil computations: Laplacian, Divergence, and Gradient
  - Implemented using central-difference on 256^3 grid
- Four test platforms: AMD Barcelona, Intel Nehalem, Sun Victoria Falls, and NVIDIA GTX 280
  - All have Flop:DRAM byte ratios >> arithmetic intensity of stencil computations, so assumed to all be memory bound
- Benchmarked against serial & OpenMP variations

Results – Laplacian



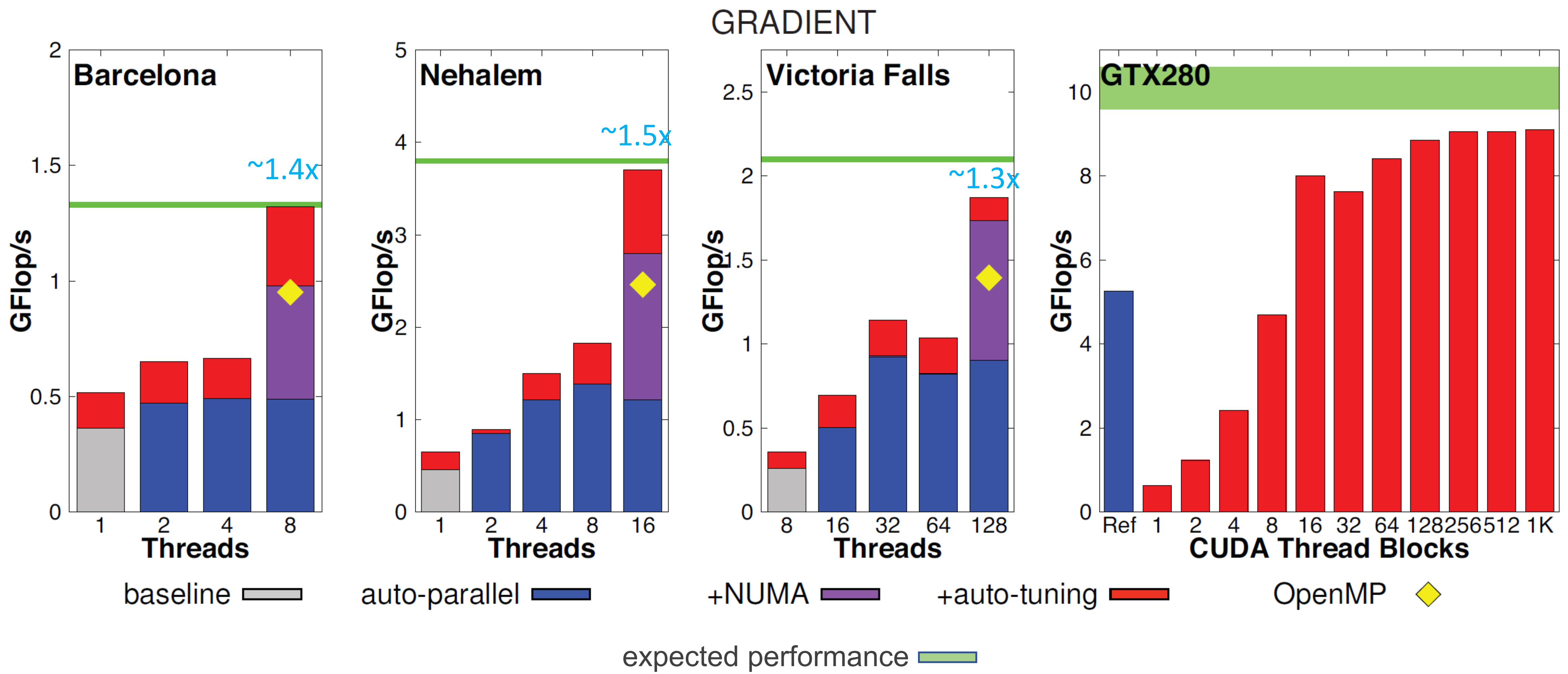
Approximate improvement vs OpenMP

Results – Divergence



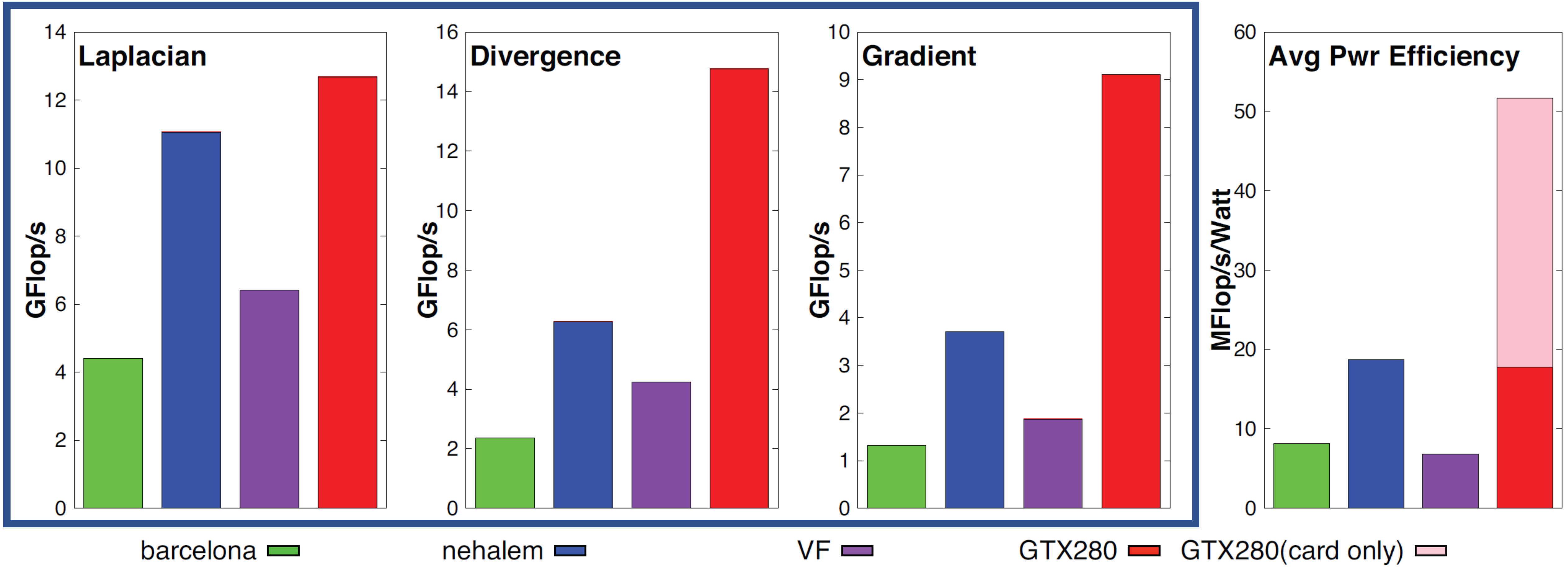
Approximate improvement vs OpenMP

Results – Gradient



Approximate improvement vs OpenMP

Results – Peak Performance & Power Efficiency



Probably out of time?

Discuss!