Study and implementation of computational methods for Differential Equations in heterogeneous systems

Asimina Vouronikoy - Eleni Zisiou

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ
2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Outline

- **Introduction**
- Review of related work
- Cyclic Reduction Algorithm
- Block Cyclic Reduction Algorithm
- Implementation
- Optimizations
- Results
- Future Plans

Introduction

- 1D/2D Poisson differential equations:

1. Discretization of the PDE domain into a grid of evenly spaced nodes

2. Discretization of the restriction of the PDE equation on the grid nodes with one of :
   - Finite Difference (FD) methods
   - Finite Element methods (FEM)
   - Finite Volume (FV) methods

3. Solution of the resulting linear system
   - Iterative (Jacobi, Gauss-Seidel, SOR, Multigrid etc)
   - Direct methods (Gauss, Cholesky, Thomas, FFT etc).

Tridiagonal solvers

- Tridiagonal solvers are tools of high importance in wide range of engineering and scientific applications
  - Applications include:
    - Computer graphics
    - Financial applications
    - Fluid dynamics
    - Modeling of medical problems
  - Various solving methods:
    - Thomas algorithm
    - Cyclic reduction method
    - Recursive doubling etc.

Contribution

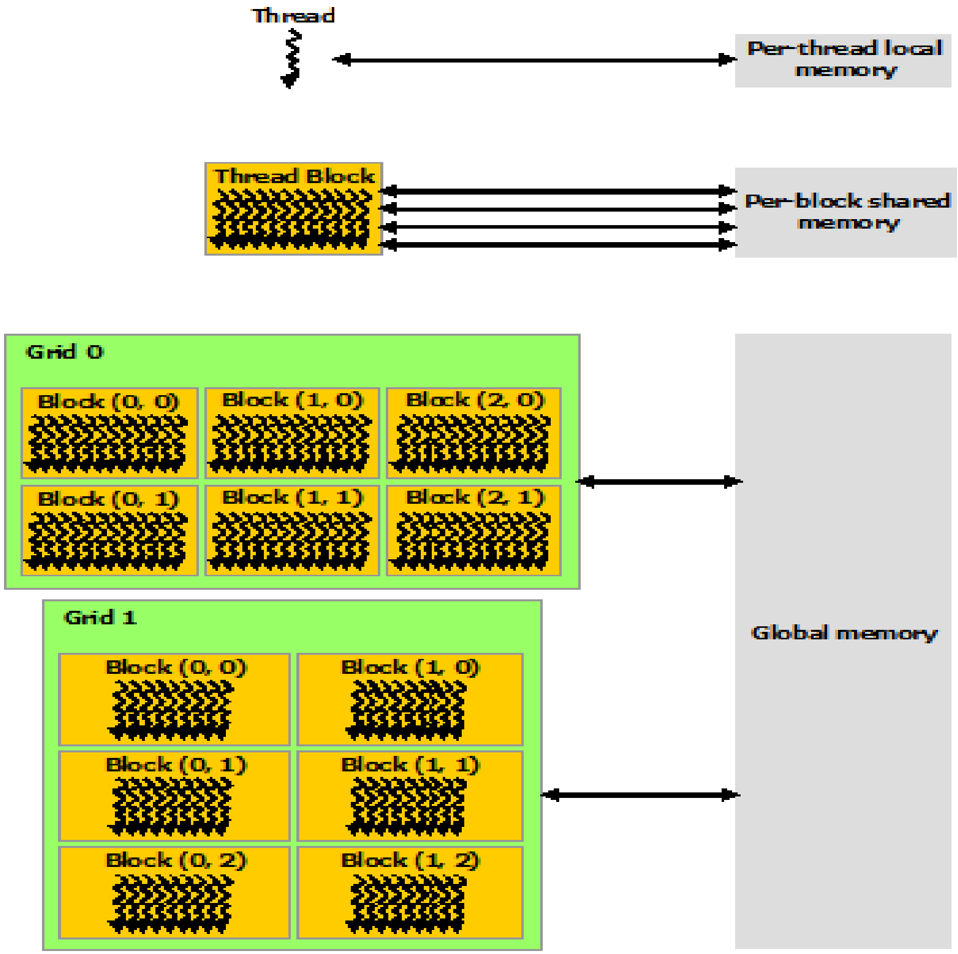
- Extremely intensive computations → need to solve very fast PDEs in 2D problems

- **Contribution**
  - Algorithms for solving
    - tridiagonal systems arising from 1D PDEs
    - and block tridiagonal systems arising from 2D PDEs
      - CPU implementation
      - GPU implementation
  - Model : Poisson differential equation in 1D and 2D

CPUs vs. GPUs

- **CPUs**
  - Few cores optimized for serial processing
  - Large caches
  - Slow context switch
  - General purpose computation
- **GPUs**
  - Thousands of smaller, more efficient cores designed for parallel performance
  - Small caches
  - Fast hardware implemented context switch
  - Need of intensive and simple operation

CUDA programming model



- Introduction
- **Review of related work**
- Cyclic Reduction Algorithm
- Block Cyclic Reduction Algorithm
- Implementation
- Optimizations
- Results
- Future Work

Review of related work

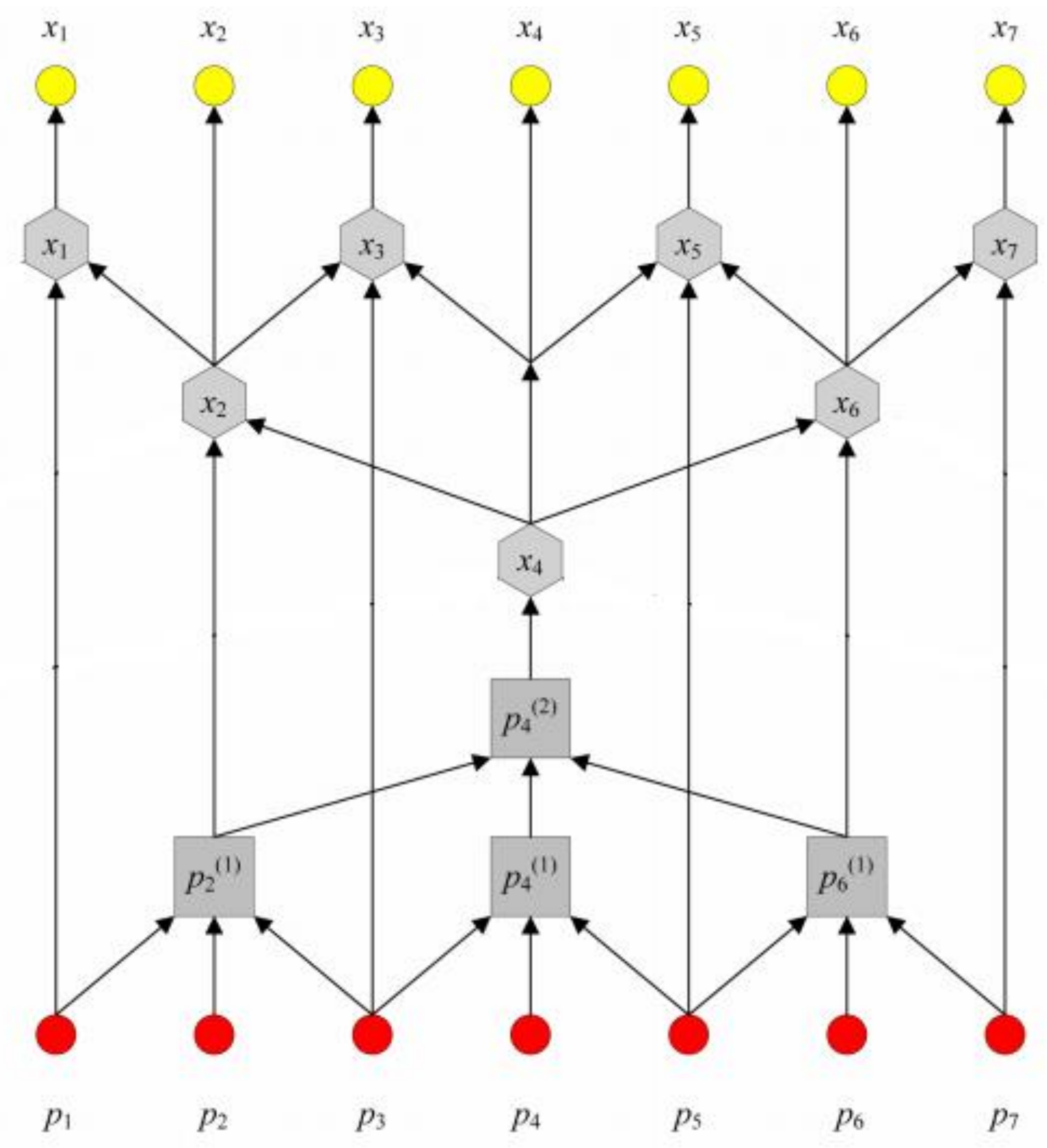
- Most popular discretization methods
  - Finite Difference
  - Finite Element
- Most popular linear solving methods
  - Parallel Cyclic Reduction
  - Conjugate Gradient
  - Jacobi
- Most compared to
  - CUBLAS library
  - CUSPARCE library
  - MKL library

- Introduction
- Review of related work
- **Cyclic Reduction Algorithm**
- Block Cyclic Reduction Algorithm
- Implementation
- Optimizations
- Results
- Future Work

Cyclic Reduction Algorithm (1/2)

- 1D problems $\longrightarrow$ tridiagonal linear system
- Two step algorithm
  - Forward reduction
    - Combine linearly the equations in order to eliminate the odd numbered unknowns
    - Unknowns are re-ordered
    - Process is continued until one equation with one unknown is left
  - Backward substitution
    - Solve the one equation left and find the unknown x
    - Find all unknowns from the previous steps
- Each phase consists of $log_2(n+1)$ steps, n = system size
- n = $2^p - 1$

Cyclic Reduction Algorithm (2/2)



- Introduction
- Review of related work
- Cyclic Reduction Algorithm
- **Block Cyclic Reduction Algorithm**
- Implementation
- Optimizations
- Results
- Future Work

Block Cyclic Reduction Algorithm (1/2)

- 2 D problems $\longrightarrow$ block tridiagonal systems

Solution of A * X = F

$$A = \begin{pmatrix} B & T & & & & \\ T & B & T & & & \\ & T & B & T & & \\ & & \cdots & & & \\ & & & \cdots & & \\ & & T & B & T & \\ & & & B & T & \end{pmatrix}$$

where B is tridiagonal matrix and T is a diagonal matrix

Block Cyclic Reduction Algorithm (2/2)

- Extension of Cyclic Reduction to block tridiagonal systems (n blocks of size q)

- After $log_2(n+1)$ reductions, a 1x1 block system needs to be solved

- Formulation is numerically unstable

$\Downarrow$

- Buneman variant

Buneman algorithm

- Buneman series (P, Q auxiliary vectors) where k= iteration

$$T^{(k)} = T^{2^k}$$
$$[B^{(-1)}]^{(k)} = -\sum_{l=1}^{2^k} a_{(kl)} * [B - 2\cos(\theta_{kl}) * T]^{(-1)}$$
$$\theta_{kl} = \left(l - \frac{1}{2}\right) * \frac{p}{2^{jq}}$$

- $$a_{kl} = (-1)^l / 2^k * \sin(\theta_{kl})$$

Initialize $q_j^{(0)} = F_j$ and $p_j^{(0)} = 0$

- Then for k = 1, .. ,jq for j=1,..., $2^{jq-k} - 1$ jq= $log_2(n+1)$

Solve $B^{(k-1)}X = p_{2j-1}^{(k-1)} + p_{2j+1}^{(k-1)} - q_j^{(k-1)}$ for X

$$p_j^{(k)} = p_j^{(k-1)} - X$$
$$q_j^{(k)} = q_{2j-1}^{(k-1)} + q_{2j+1}^{(k-1)} - 2p_j^{(k)}$$

- Introduction
- Review of related work
- Cyclic Reduction Algorithm
- Block Cyclic Reduction Algorithm
- **Implementation**
- Optimizations
- Results
- Future Work

Implementation issues – CR

- Storage demands: 5 vectors
  - 3 diagonals
  - 1 rhs vector
  - 1 solution vector

- Data dependencies between iterations
  - Both in Forward Reduction and Backward Substitution phase

Implementation issues – Buneman

- High storage demands

- In every forward reduction step, matrices B and T are modified
  - Must be stored for the backward phase

- Possible solutions :
  - Store them
  - Recalculate them in every step

- Introduction
- Review of related work
- Cyclic Reduction Algorithm
- Block Cyclic Reduction Algorithm
- Implementation
- **Optimizations**
- Results
- Future Work

Optimizations - CR

- Dynamic calculation of the block dimension
  - The geometry changes while the size of the system is growing
- Padding
  - Eliminate the if – branches in Backward Substitution kernel

Optimizations - Buneman (1/2)

- CPU implementation
  - MKL & LAPACK libraries

- GPU implementation
  - CUBLAS & CULAPACK libraries

- Routines for the BLAS operations
    - Matrix-Matrix Multiplication
    - Matrix-Vector Multiplication
    - Inverse Matrix
    - Matrix addition
    - Vector addition
    - Scalar Matrix Multiplication

Optimizations - Buneman (2/2)

- Restructure the code by merging math operations to use optimally the libraries

- Avoid the inversion of matrix B by solving a linear system

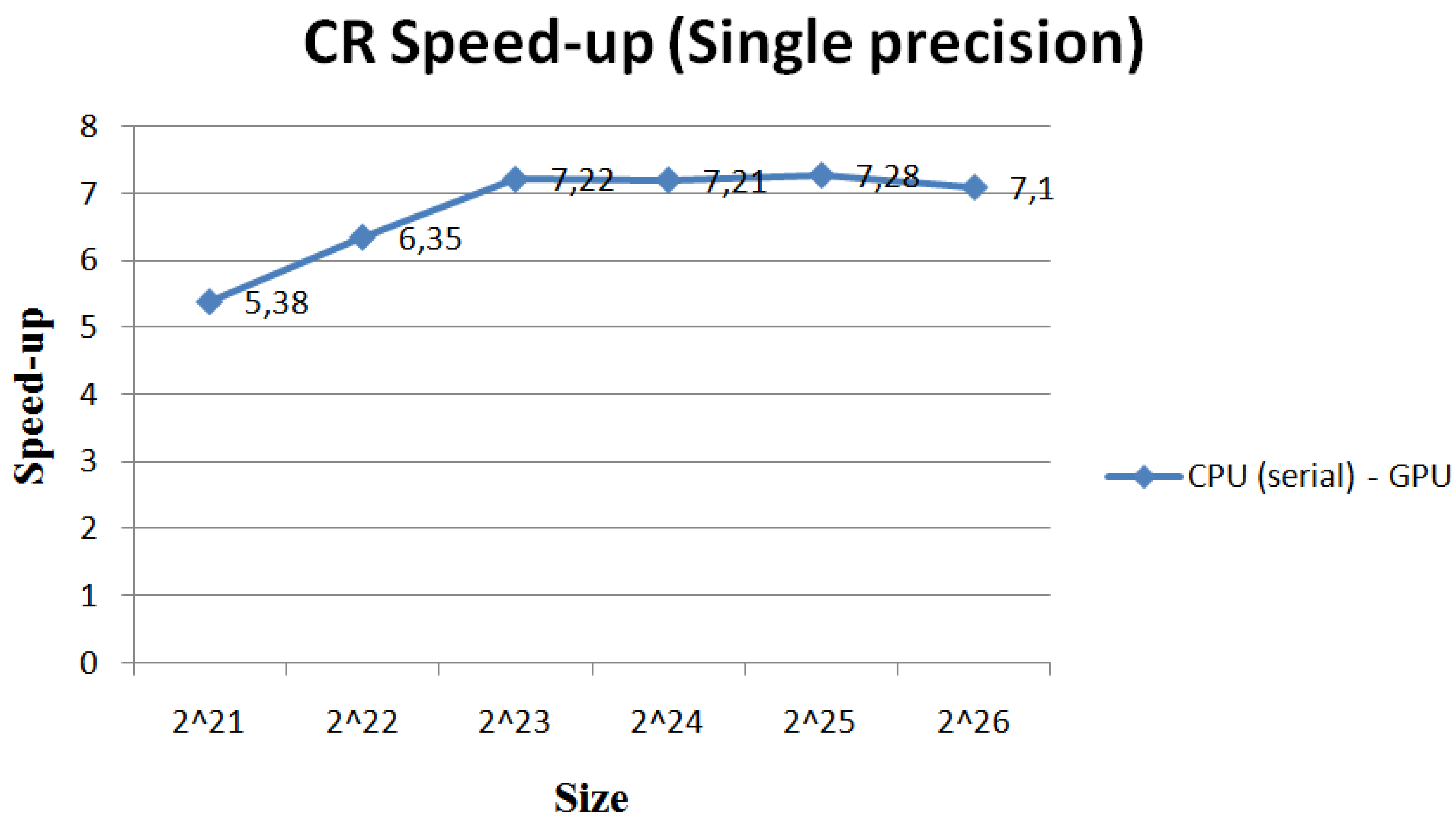
- ”Sliding window” technique to examine larger problems
  - Asynchronous memory transfers

- Introduction
- Review of related work
- Cyclic Reduction Algorithm
- Block Cyclic Reduction Algorithm
- Implementation
- Optimizations
- **Results**
- Future Work

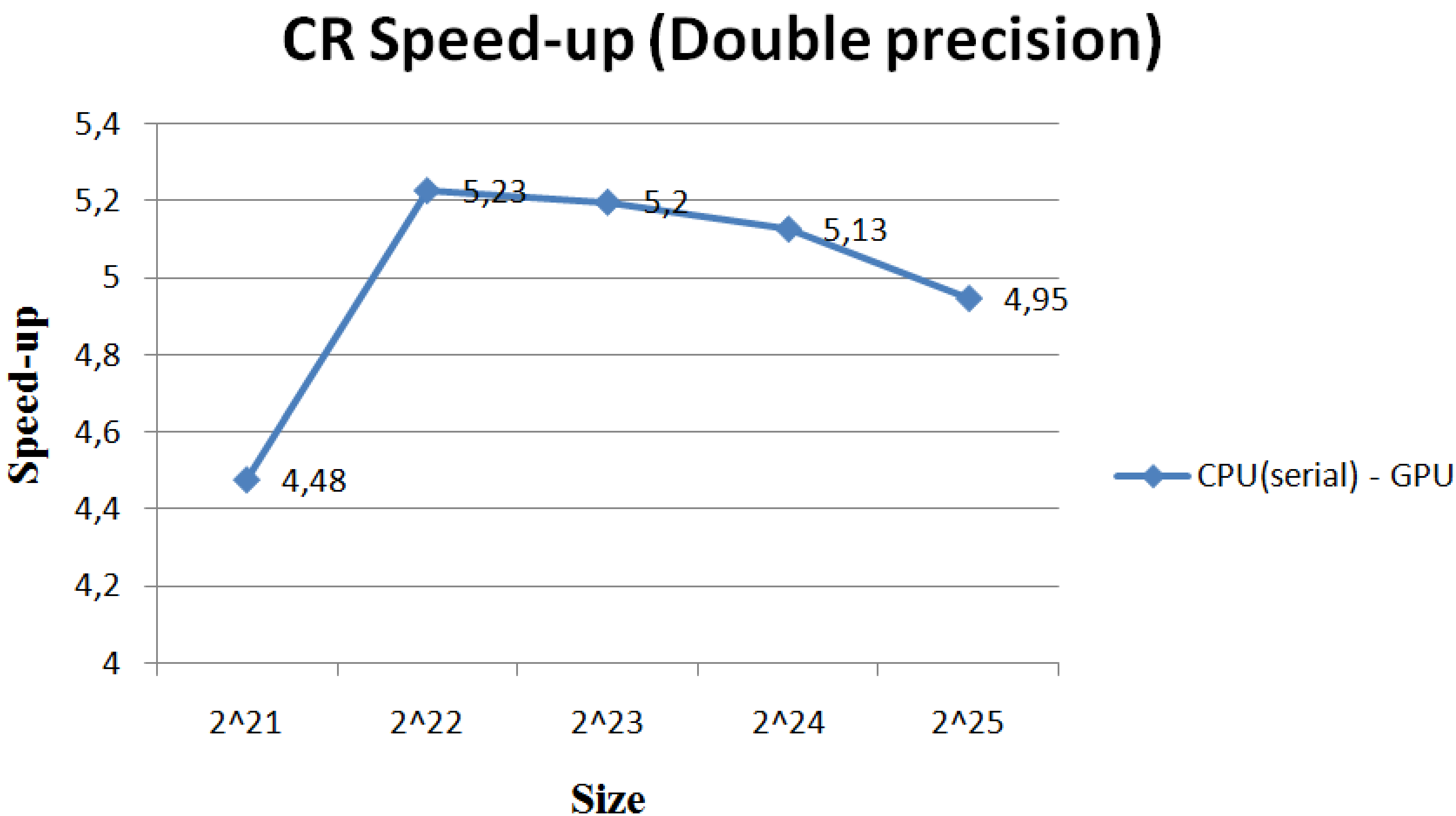
Experiments

- The hardware used for the experiments is :
    - Intel Xeon CPU W3550 @3.07 GHz with 8GB RAM and four cores
    - NVIDIA GeForce GTX680
- clock() function , time presented in seconds
- Full optimizations turned on
- CPU: icc compiler
- GPU: nvcc compiler
- CR
    - Sequential CPU implementation
    - GPU implementation
- BCR
    - Sequential CPU implementation
    - Parallel CPU implementation
    - GPU implementation

Results - CR

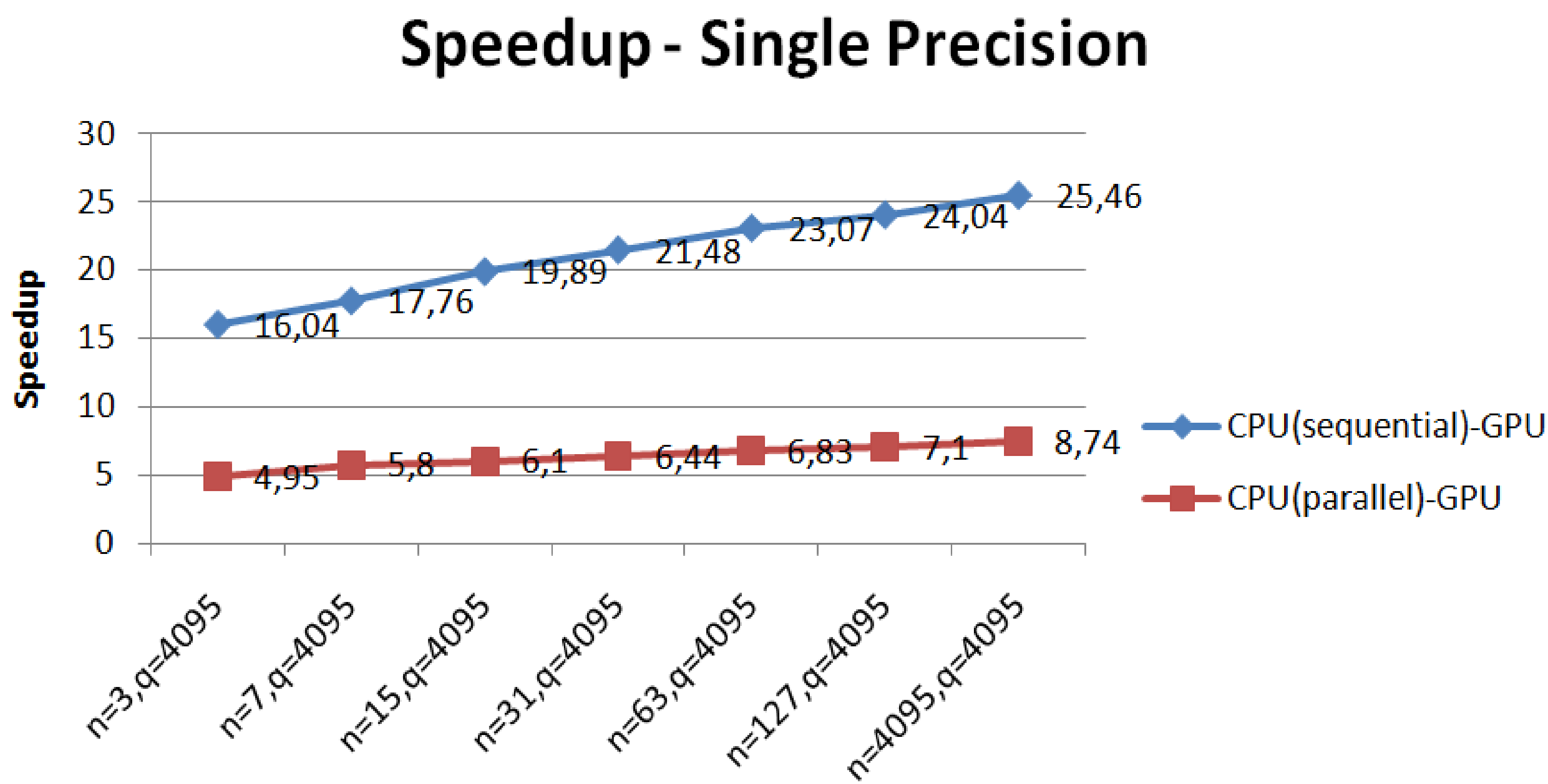

CR Speed-up (Single precision)
Speed-up
8
7
6
5
4
3
2
1
0
5,38
6,35
7,22
7,21
7,28
7,1
2^21
2^22
2^23
2^24
2^25
2^26
Size
CPU (serial) - GPU

Results - CR



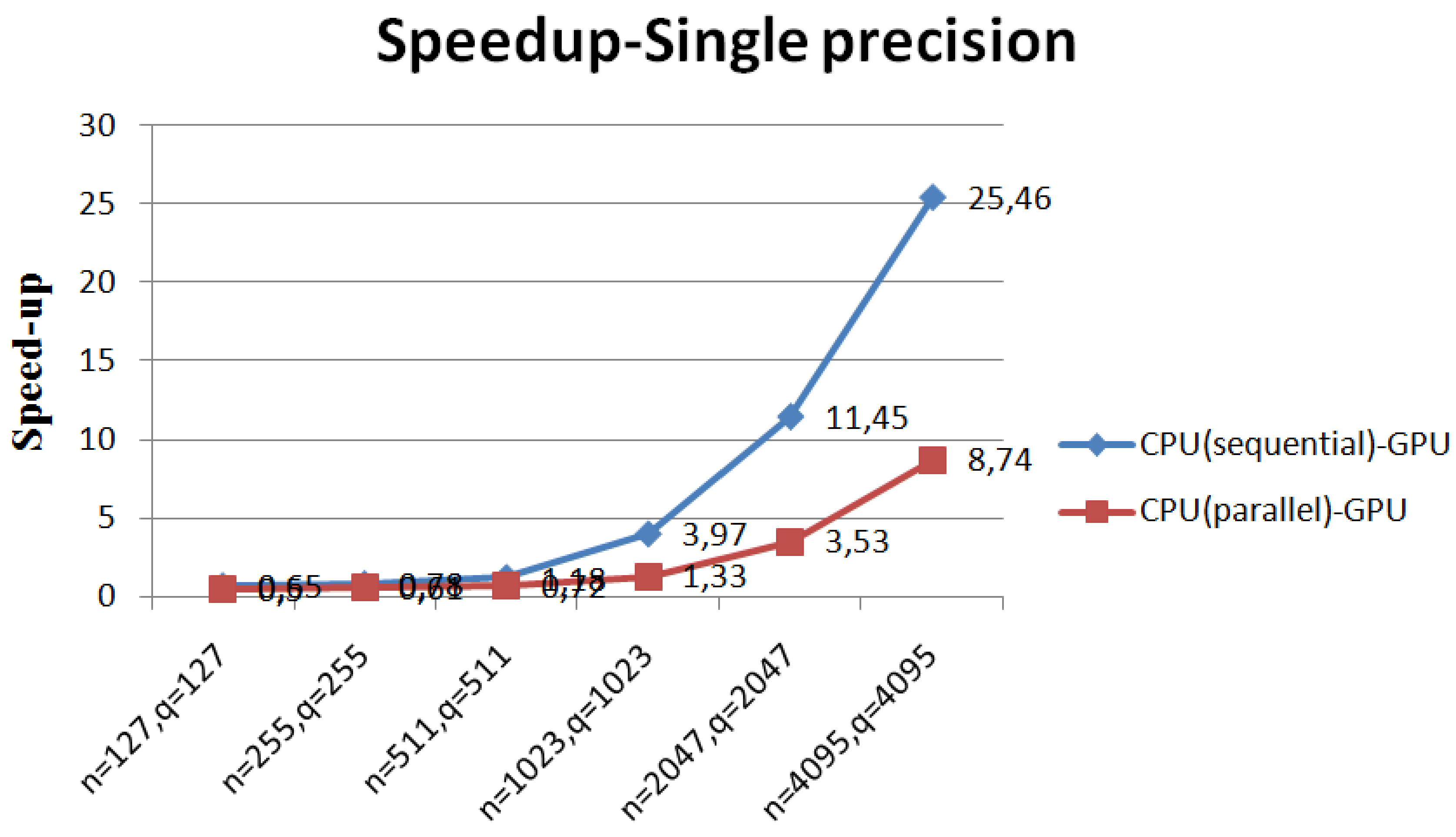
CR Speed-up (Double precision)
5,4
5,2
5
4,8
4,6
4,4
4,2
4
Speed-up
4,48
5,23
5,2
5,13
4,95
2^21
2^22
2^23
2^24
2^25
Size
CPU(serial) - GPU

Results – Buneman

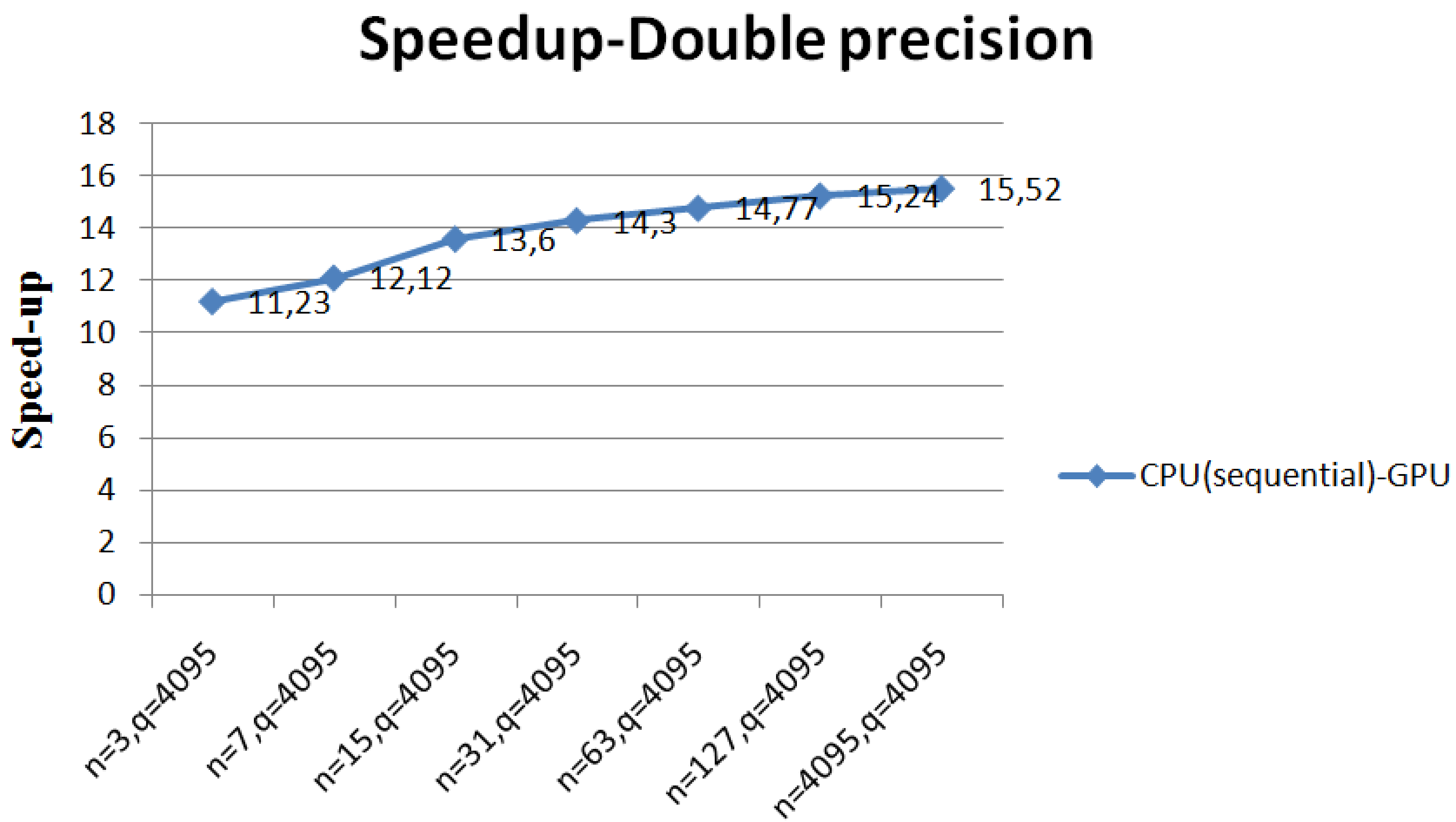


- GPU results shown above arise from “sliding window” version . Performance from first version is < 1% different . CPU results arise from “inverse” version. Performance from “solver” version is the same.

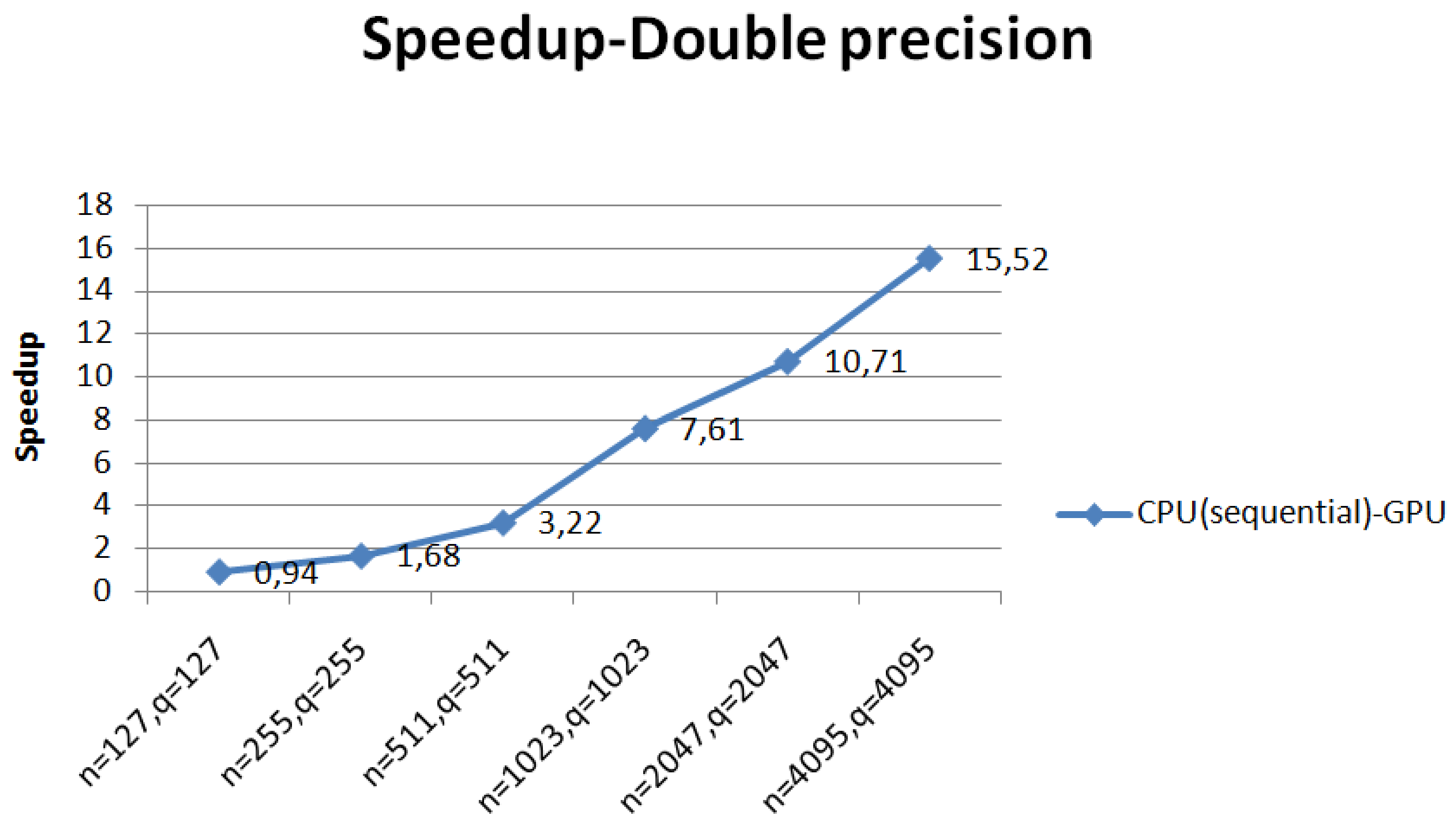
Results – Buneman



Results – Buneman



Results – Buneman



- Introduction
- Review of related work
- Cyclic Reduction Algorithm
- Block Cyclic Reduction Algorithm
- Implementation
- Optimizations
- Results
- **Future Work**

Future Work

- As a case study we plan to apply Buneman method to a reaction-diffusion PDE

- This PDE is considered as a common simplified model for studying the expansion of gliomata

Ευχαριστούμε!

Ερωτήσεις;