Speeding Up *ChemApp* by Parallelisation

Khan, Wajahat Murtuza

Master's Student,
Software Systems Engineering,
Aachen University of Technology (RWTH)
Aachen
Germany

Werkstudent,
GTT-Technologies,
Technologiepark
Herzogenrath
Germany

GTT –Technologies' Annual Workshop
Herzogenrath, Germany. June 20 – 22, 2007

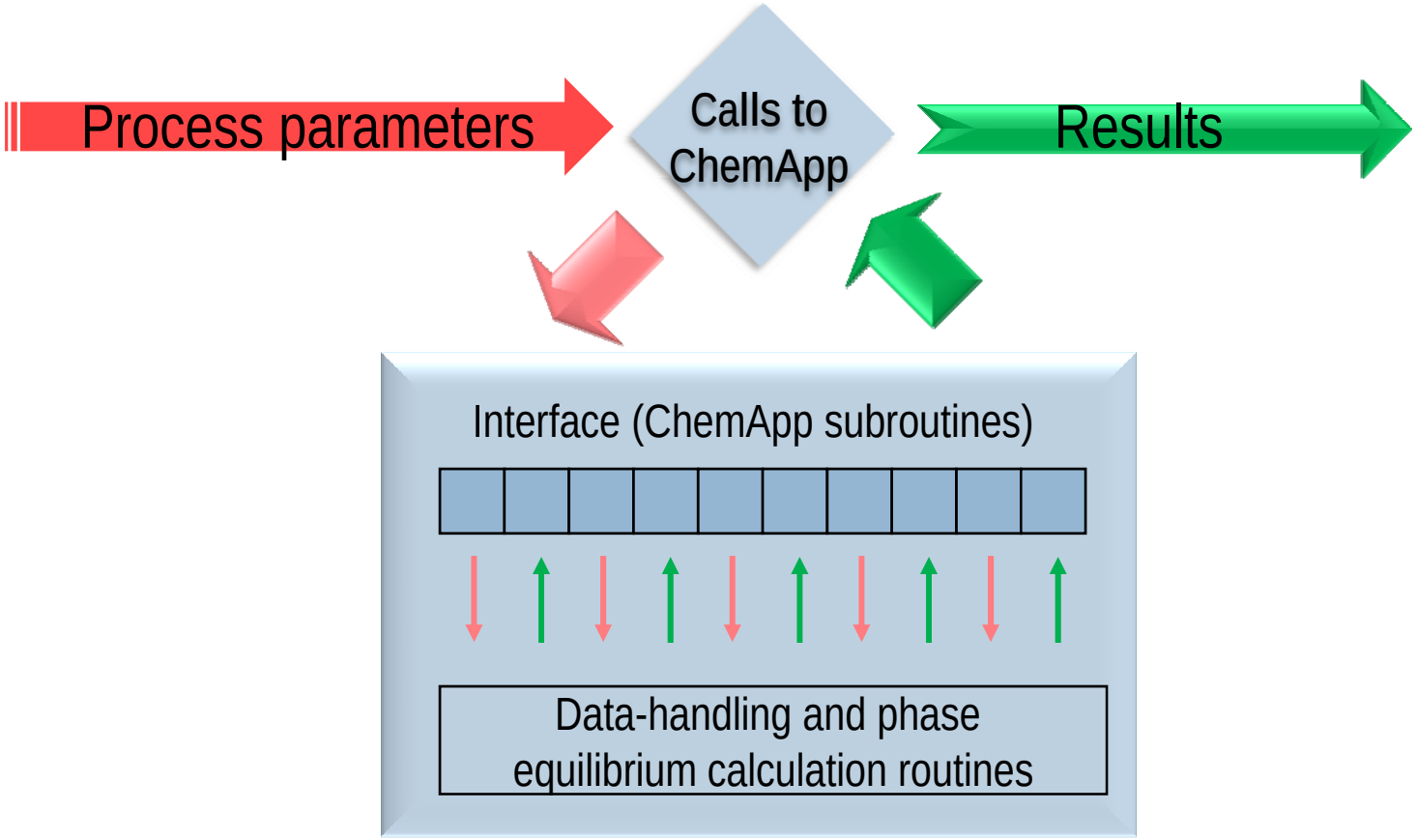
Topics

I. ChemApp Introduction
II. Parallelization and ChemApp
III. Simple Test Case
IV. Performance
V. Conclusion

ChemApp : Introduction

- A programming library consisting of rich set of subroutines that provide all the necessary tools
- ChemApp calculates :
  - Complex multi-component, multi-phase chemical equilibria
  - Determination of the associated extensive property balances.
- Available for wide range of platforms as object code and as a Dynamic Link Library
  - Fortran, C, C++, Visual Basic or Delphi

ChemApp: Schematic View



ChemApp

... is partially contained in other GTT products

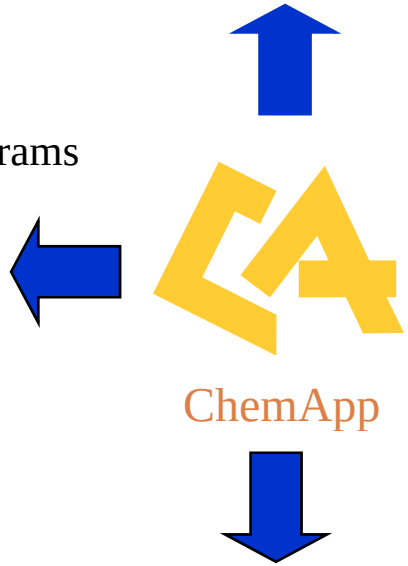
ChemSage FactSage

... is a module of 3rd party programs

CFX
Fluent
Aspen Plus

... is used in custom programs

SteelMap (GTT)
Customer´s programs



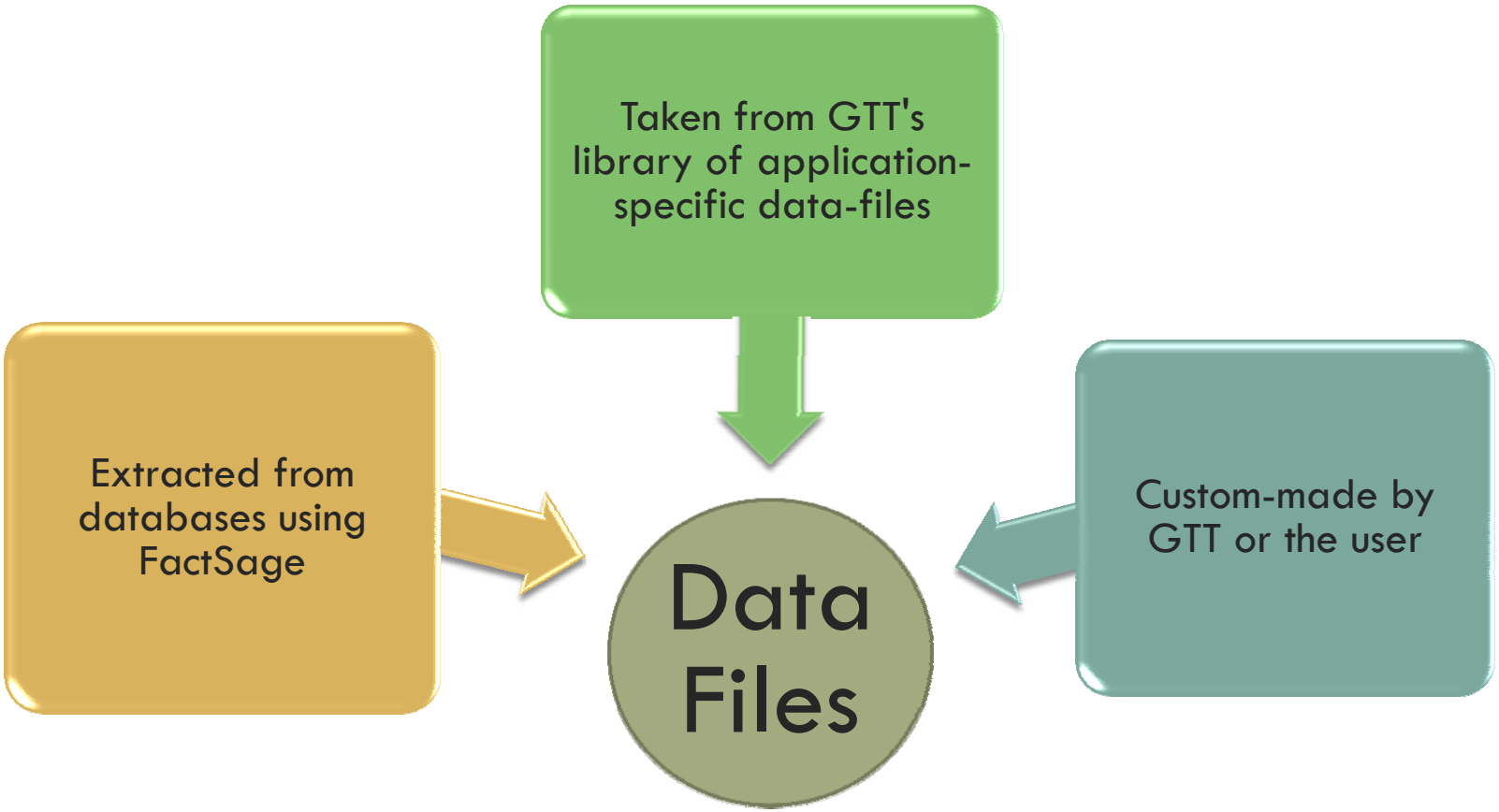
... is wholly contained in other GTT products

ChemSheet SimuSage

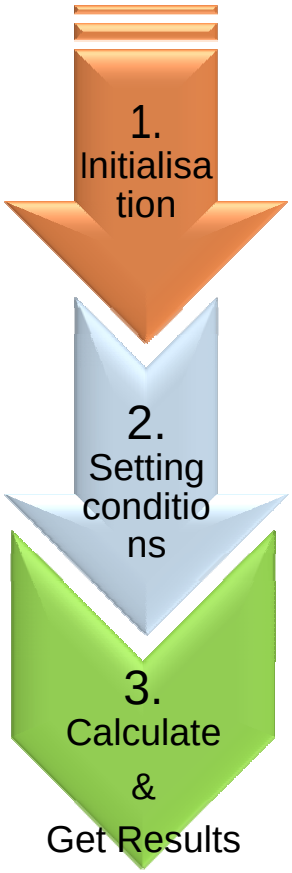
ChemApp Platforms

PC/DOS	• Lahey Fortran (LF77/LF90/LF95), Watcom Fortran and C/C++
PC/Windows	• Microsoft Powerstation Fortran, Visual Basic, Visual C++, Digital/Compaq Visual Fortran, Lahey Fortran L90/LF95, Borland Delphi, Watcom Fortran and C/C++,
Unix WS	• Linux-PCs, SGI, Sun Solaris, HP, IBM, Sun, Cray, DEC Alpha...

Sources of data-files



ChemApp : 3 Steps



- ***Initialize*** *the interface*
- *Read a thermodynamic data file*
- *Adjust the chemical system*

- ***Set initial conditions*** *for the equilibrium calculations*

- *Perform the* ***calculations***
- *Get* ***results***

ChemApp : 1st Step

1. *Initialize the interface, read a thermo-dynamic data file,adjust the chemical system*

- A subroutine call to initialize ChemApp : (***tqini***)
- A call to read the data-file (***tqopen***, ***tqrfil***, ***tqclos***)
  - *The thermochemical data for the observed system is loaded from a separate data-file which is completely independent from the ChemApp code*
- Adjust the units: (***tqsetc***)
  - *Pressure - bar, atm, Pa, kPa, psi, torr*
  - *Volume - dm3, cm3, m3, ft3, in3*
  - *Temperature - Kelvin, Celsius, Fahrenheit*
  - *Energy - J, cal, Btu, kWh*
  - *Amount - Mol, gram, kg, tonne, pound*
- Change the status of phases, if desired (***tqcsp***)
  - *Entered, Dormant, Eliminated*

ChemApp : 2nd Step

2. Set Initial Conditions for equilibrium calculations

- Two different methods available for defining initial conditions:

i. **Global conditions**: Specification of pressure, temperature, and incoming amounts of substances. (***tqsetc***)
   - Result: thermochemical equilibrium

ii. **Streams**: Specification of pressure, temperature, definition of incoming amounts as non-reacted mixture with constant temperature and pressure. (***tqstec***)
   - Result: thermochemical equilibrium, extensive property (e.g. heat) balance

ChemApp : 3rd Step

3. *Perform the calculation and get results*

- Only one routine needs to be called to calculate the chemical equilibrium (**tqce**).
- Results can be obtained (**tqgetr**) for the following variables:
  - Total pressure, total volume, temperature
  - Equilibrium amount of phases, phase constituents, and system components
  - Chemical potential and activity
  - Heat capacity, enthalpy, entropy, and Gibbs energy of the equilibrium state
  - Mass or mole fraction of a system component or phase constituent
- ChemApp can also calculate the thermodynamic properties; Cp, H, S, and G, of a single phase and/or its constituents.
- Perform one-dimensional phase mapping calculations

II

Parallelization and *ChemApp*

What is Parallelization?

Multiple processors for a single task

- The use of multiple computers or processors working together on a common task

Parallelized *ChemApp*

- Faster calculation of thermochemical equilibria on a multicore processor

Parallelization

- An optimization technique to get work done faster
- Distributing a sequential task across multiple processors and or nodes and collecting results
- Simultaneous use of multiple computing resources
- Reduces wall-clock time, solves bigger problems
- Parallelism can be at a coarser or finer level
- Parallelism can be on a shared memory or a distributed memory architecture

Motivation for Parallelization

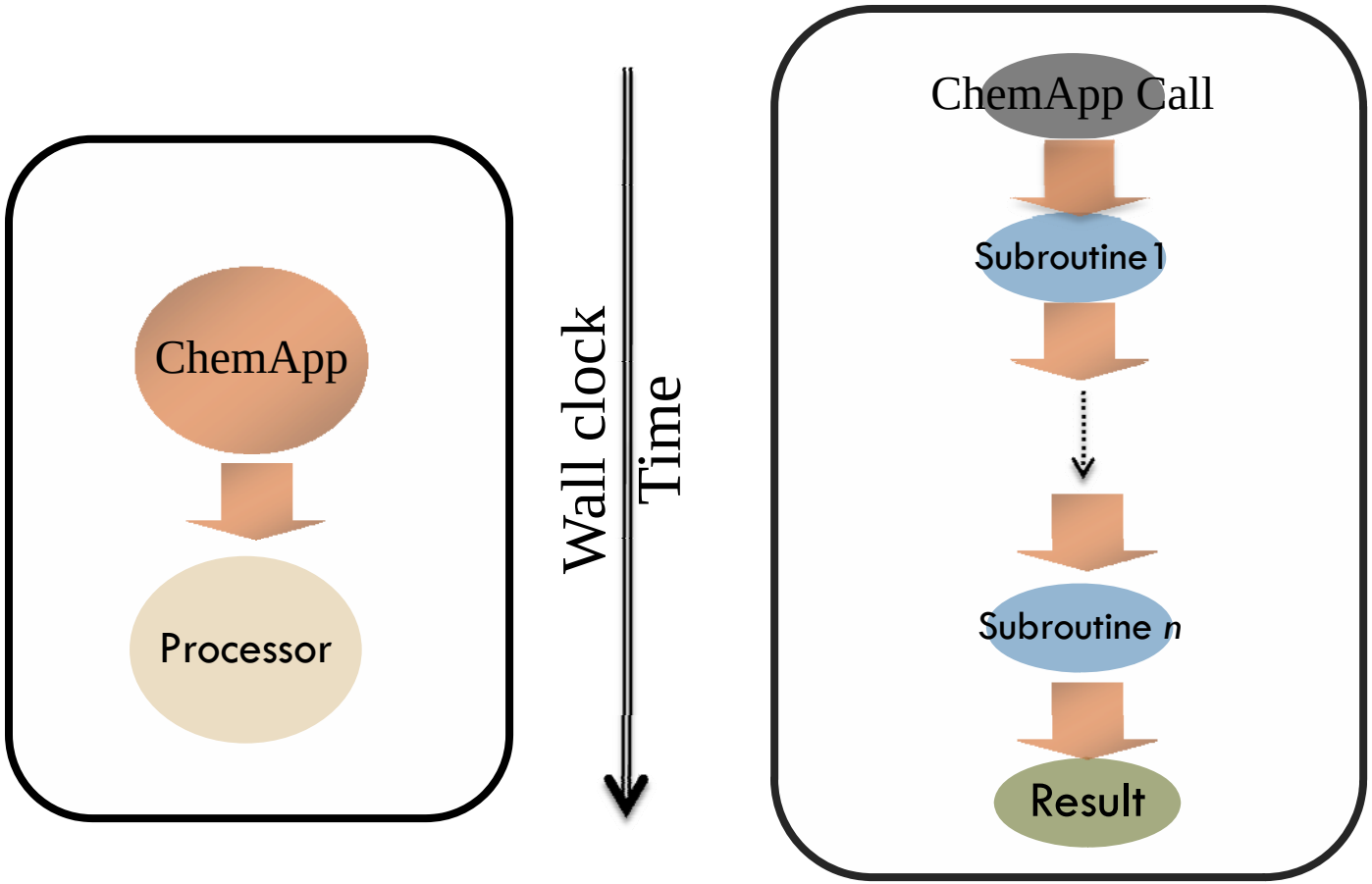
Why do parallel computing ?

- Limits of single CPU computation
  - Performance limitations
  - Processor clock speed
  - Memory
- Parallel computing allows:
  - Solving problems in reasonable time
  - Larger problems more faster
  - Realistic simulations at finer resolutions

Serial Execution

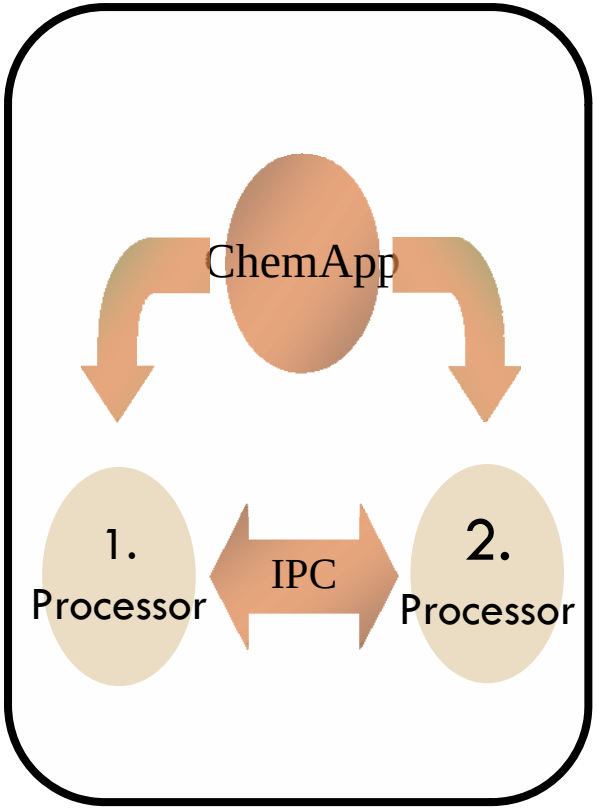
Serial Task

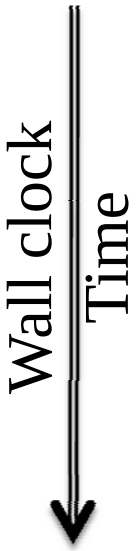
Single Thread



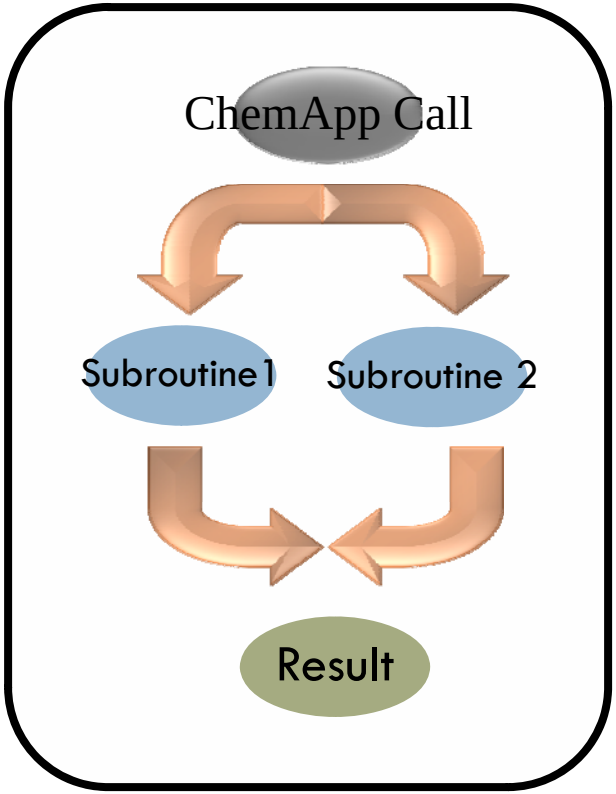
Parallel Execution

Parallel Task





Multi Thread



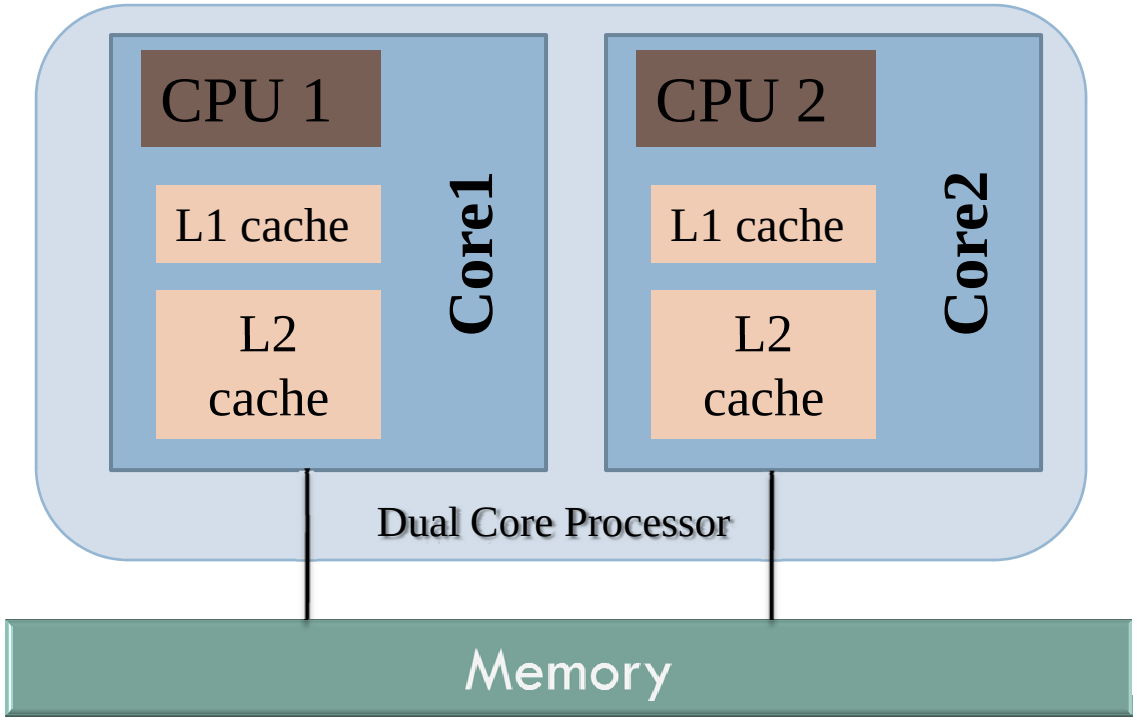
Parallel Programming Models

Models classified according to memory access

- Distributed Memory Architecture
  - Memory ***distributed*** across various nodes
    - *PVM*: Parallel Virtual Machine (obsolete)
    - *MPI*: Message Passing Interface (de-facto standard)
- Shared Memory Architecture
  - Memory available ***locally*** to various processors
    - *Posix* threads (very low-level)
    - *OpenMP* (de-facto standard, multiple levels)

Shared Memory Model

A typical multicore processor on a single computer represents a shared memory model



Why OpenMP ?

Parallel computers are ubiquitous !

- Multicore processors are standard in today's computers
- Multicore processors from major vendors
  - *Intel*: Pentium D, Core Duo, Core2Duo, ...
  - *AMD*: Opteron dual core, Opteron quad core, Athlon X2, ...
  - *Sun*: Ultrasparc IV, Ultrasparc IV+, Ultrasparc T1, ...
- OpenMP supported by increasing number of compilers

III

A Simple Test Case

Calculating equilibrium of a multi component system

Simple Test Case

- Computation of thermochemical equilibrium of a multi-component system
  - Mo – Cu – Ni – Fe – Cr – C
  - 26 phases
  - Total 184 Phase constituents
- Varying incoming amounts of Cr and Fe by 0.01
  - Cr varies between 0.1 mol to 0.45 mol
  - Fe varies between 0.55 mol to 0.9 mol
- Constant amounts of
  - Mo = 0.025 mol
  - Cu = 0.01 mol
  - Ni = 0.06 mol
  - C = 0.015 mol
  - ∑ **= 0.11mol**
- Varying temperature range between
  - 600° K to 1600° K in increments of 10° K

Simple Test Case

#	Phases	Phase Constituents	#	Phases	Phase Constituents
1	CEMENTITE	4	14	MONI_DELTA	12
2	M23C6	12	15	KSI_CARBIDE	3
3	M3C2	2	16	SIGMA	16
4	M6C	8	17	Fe-LIQUID	6
5	M7C3	4	18	MC_ETA	2
6	FCC_A1 : Me(C,N)#1	10	19	MU_PHASE	12
7	FCC_A1 : Me(C,N)#2	10	20	HCP_A3 : Me2(C,N)#1	10
8	FCC_A1 : Me(C,N)#3	10	21	HCP_A3 : Me2(C,N)#2	10
9	BCC_A2#1	10	22	C_c<graphite>(s)	1
10	BCC_A2#2	10	23	Cr3C2_m3c2(s)	1
11	R_PHASE	12	24	MoC_mc_shp(s)	1
12	LAVES_PHASE	4	25	MoNi3_moni3_gamma(s)	1
13	P_PHASE	12	26	MoNi4_moni4_beta(s)	1

Simple Test Case

- Varying incoming amounts of Cr and Fe by 0.01such that
  - ∑ (Cr, Fe) = 1 – ∑(C, Cu, Ni, Mo)
  - Cr varies between 0.1 mol to 0.45 mol
  - Fe varies between 0.44 mol to 0.79 mol

```
// Lines of Code
for ( Cr = 0.1, Fe = 0.79; Cr < = 0.45 ; Cr + = 0.01, Fe = 0.89 – Cr )
{
        // Some lines of code
        // tqsetc, a subroutine to set temperature equilibrium calculations
        tqsetc ( "ia ", 0, 5, Cr, &numcon, &noerr );
        tqsetc ( "ia ", 0, 4, Fe, &numcon, &noerr );
        // Some lines of code
}
```

Simple Test Case

- For the varying amounts of Cr and Fe,
- Temperature varies in increments of 100 between
  - 600 ° K to 1500 ° K

```
// Lines of Code
// Varying the temperature between 600 and 1600K in increments of 10
for ( temp = 600; temp < =1600; temp += 10 )
{
        // Some lines of code
        // tqsetc, a subroutine to set temperature equilibrium calculations
        tqsetc ( "T ", 0, 0, temp, &numcon, &noerr );
        // Some lines of code
}
```

Simple Test Case

- Simple call to the function **tqce** to calculate equilibrium

```
// Lines of Code
for ( Cr = 0.1, Fe = 0.79; Cr < = 0.45 ; Cr + = 0.01, Fe = 0.89 – Cr )
{
        // Some lines of code
        for ( temp = 600; temp < =1600; temp += 10 )
        {
                // Some lines of code
                tqsetc ( "T ", 0, 0, temp, &numcon, &noerr );

                // A single call to tqce calculates equilibrium
                tqce ( " " , 0, 0, darray2, &noerr );
                // Some lines of code
        }
}
```

Simple Test Case

Number of times equilibrium calculated

- Function ***tqce*** called multiple times within the nested loop of varying amounts of Cr and Fe and the loop of varying temperatures
  - # of iterations for varying amounts of Cr and Fe = ***45***
  - # of iterations of varying temperature = **100**
  - # of times tqce called : 45 x 100 = **4500**

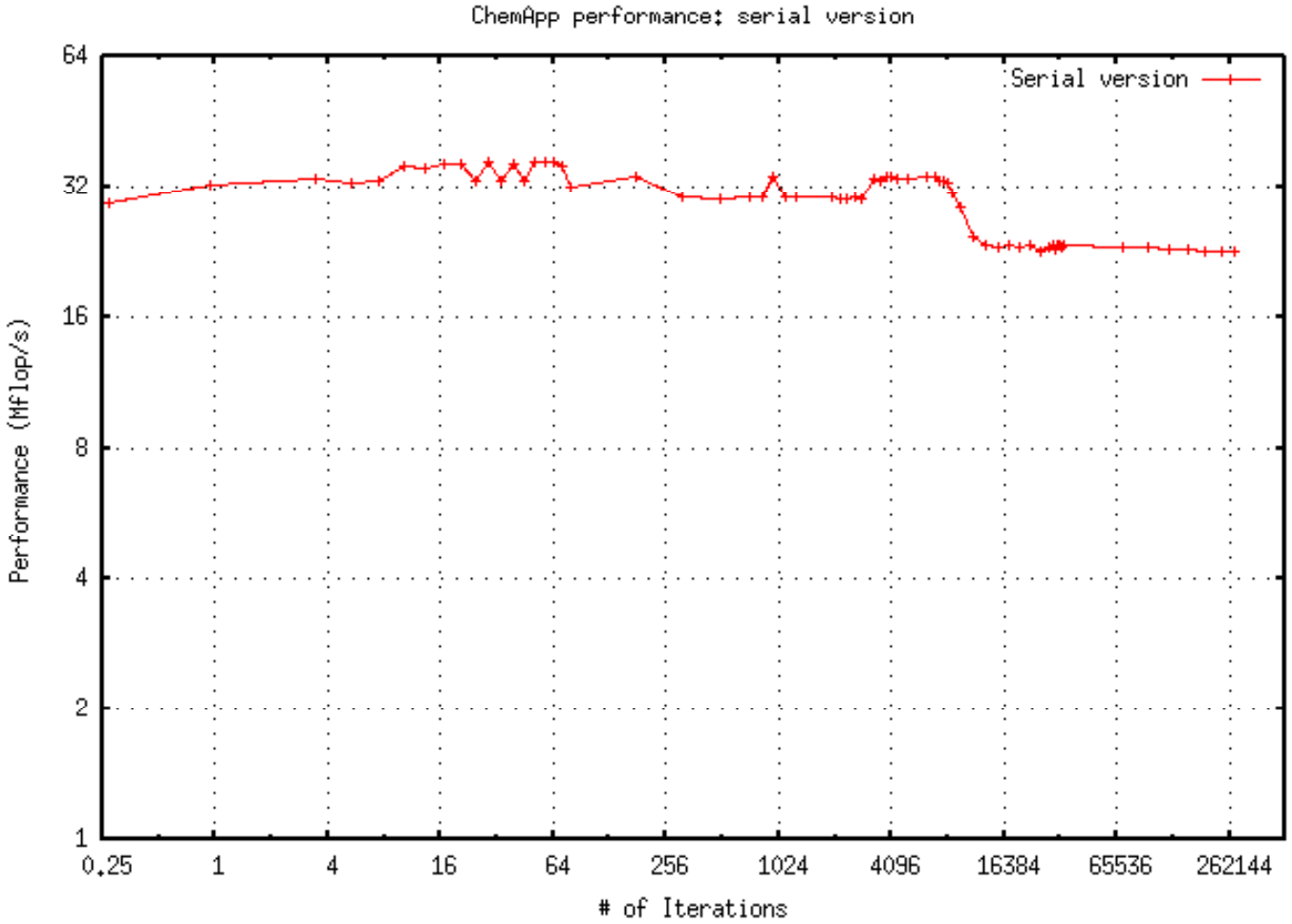
Serial Version

Performance of Serial Version of ChemApp

A few program performance metrics

- Time taken = 1020.7 seconds
- Peak performance = 40 Mflops
- CPU = Opteron 2.2 Ghz Dual Core
- RAM = 4 GB
- OS = Linux

Serial Performance



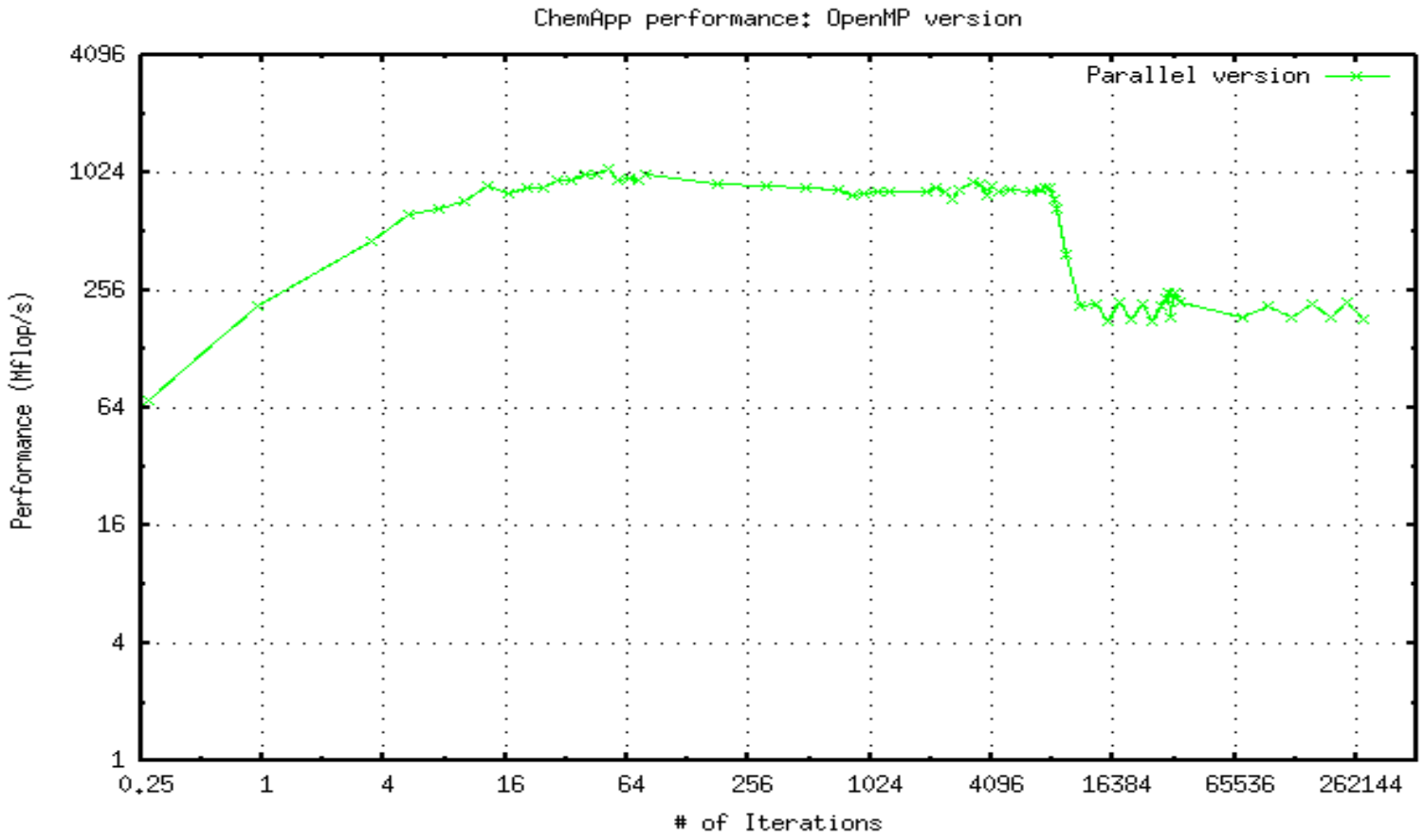
Parallel Version

Performance of Parallel Version of ChemApp

A few program performance metrics

- Time taken = 25.7 seconds
- Peak performance = 1050 Mflops
- # of parallel threads = 4
- CPU = Opteron 2.2 Ghz Dual Core
- RAM = 4 GB
- OS = Linux

Parallel Performance



Performance Enhancement

Serial Version

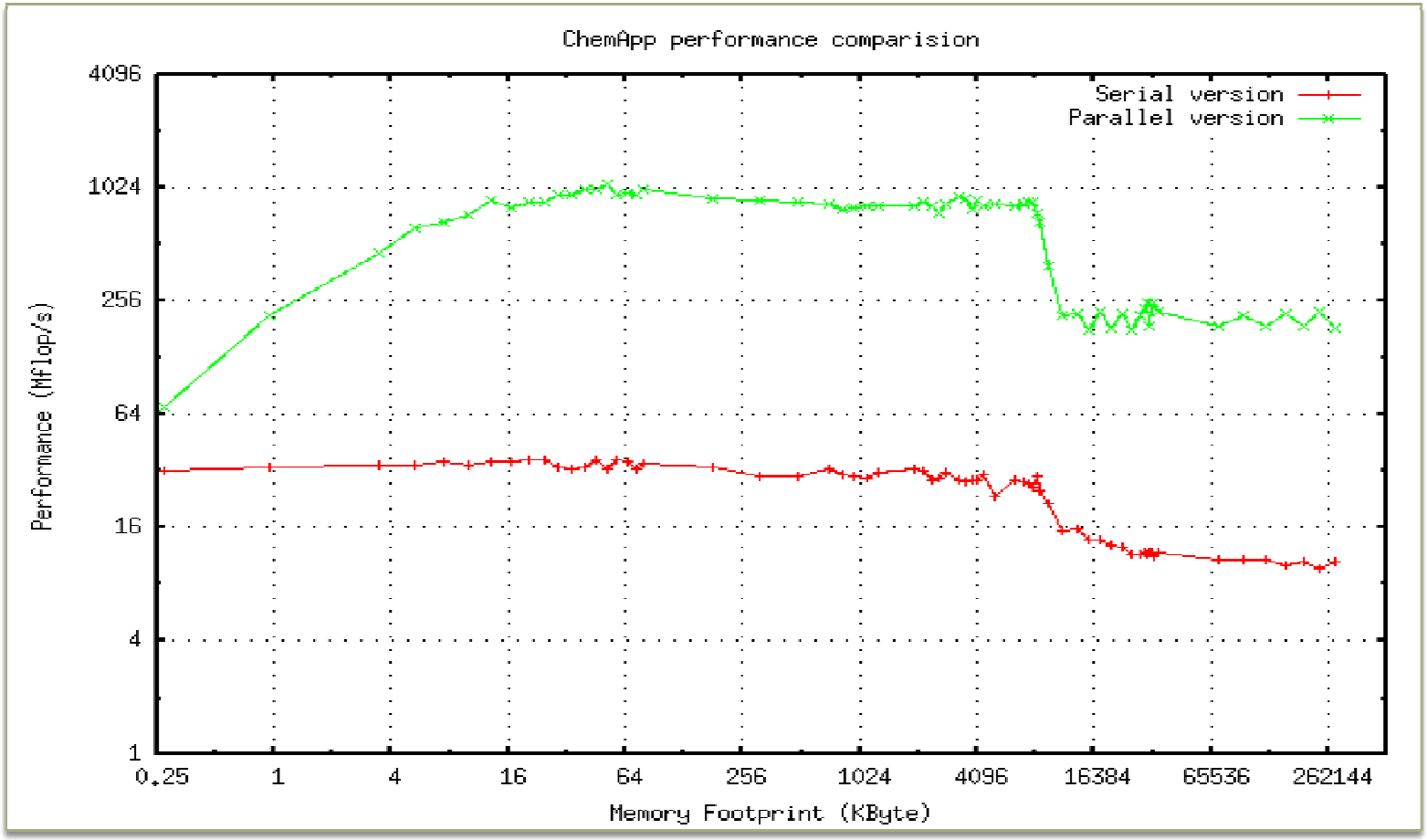
- Time taken = 1020.7 seconds
- Peak performance: 40 Mflops

Parallel Version

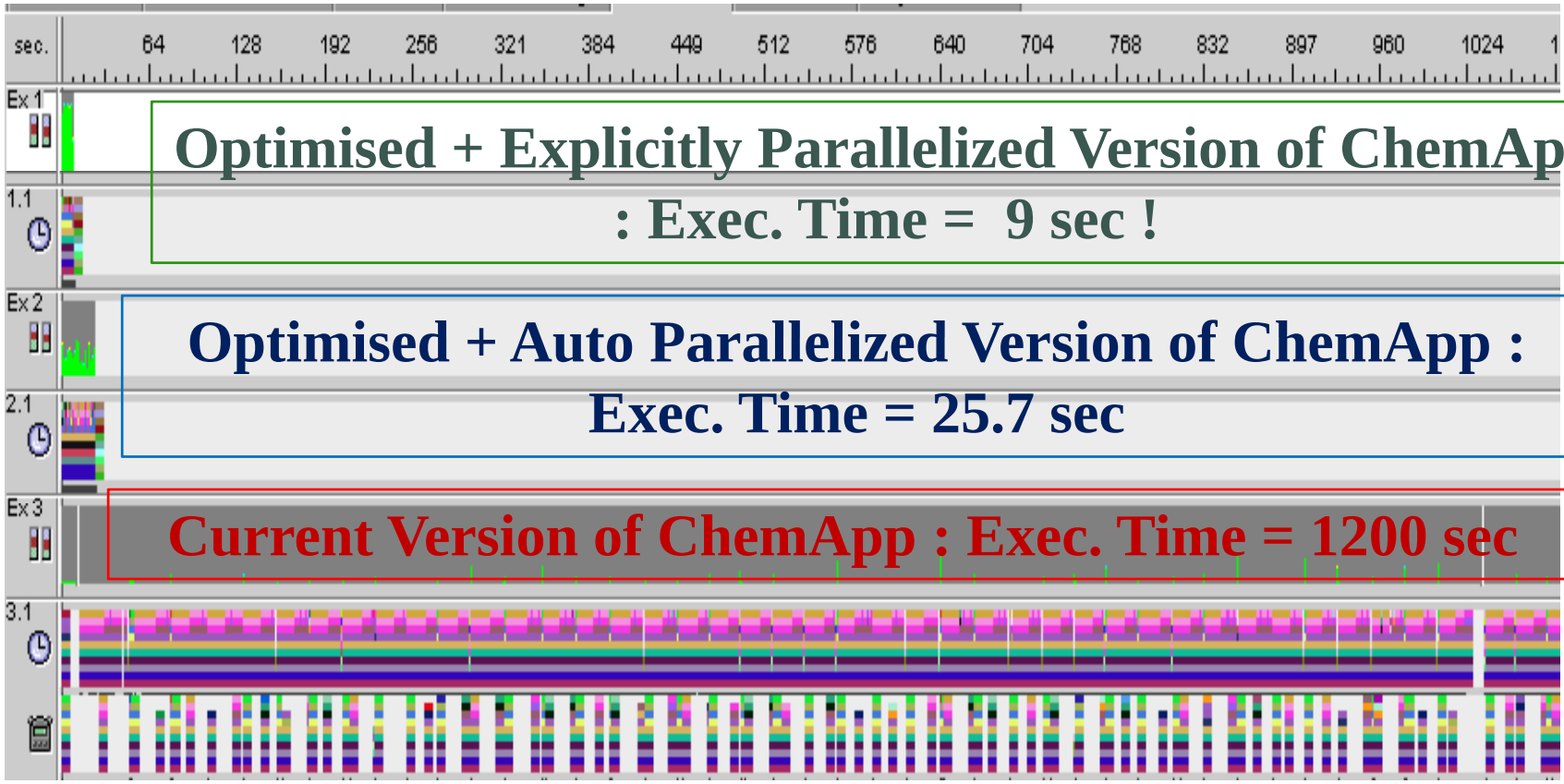
- Time taken = 25.7 seconds
- Peak performance:1050 Mflops

Execution time reduced by 7.28 times
Average performance increased by 26.25 times

Performance Enhancement



Performance Enhancement

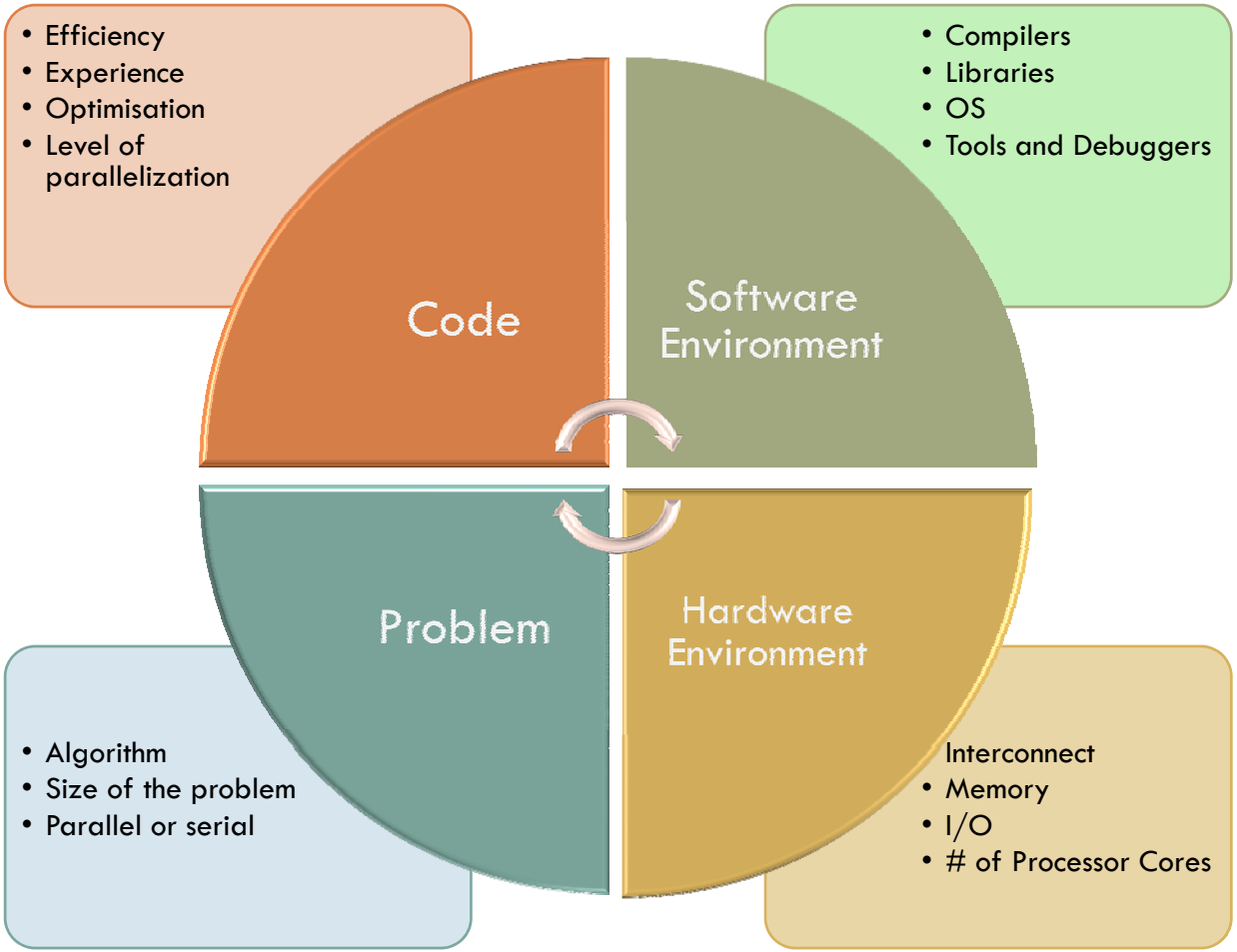


Performance Factors

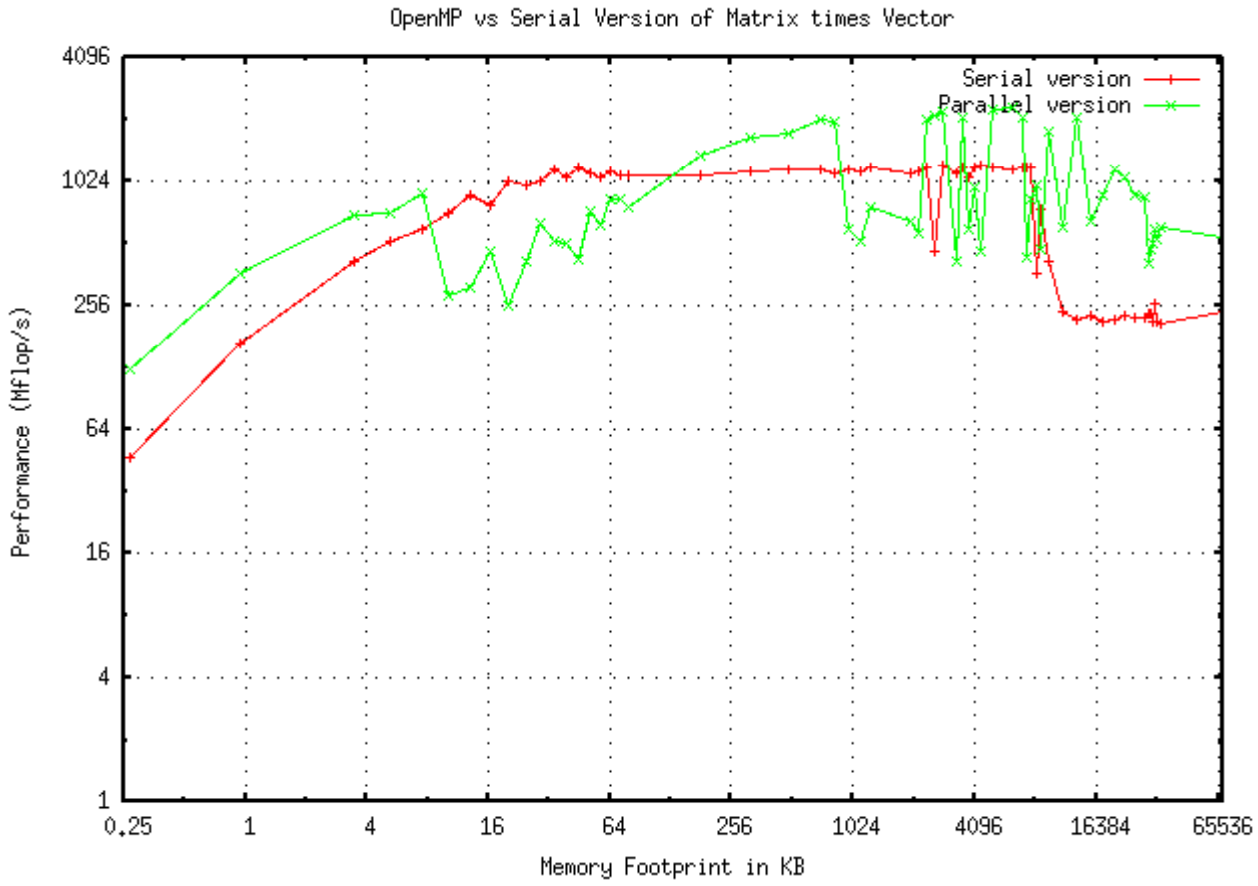
Is performance always enhanced?

Not necessarily!

Performance Factors



Performance : Serial and Parallel



Conclusion

Limitations of Parallelization

For a parallel program

- One processor gives results in N hours
- Why not use N processors
  -- and get the results in just one hour ?

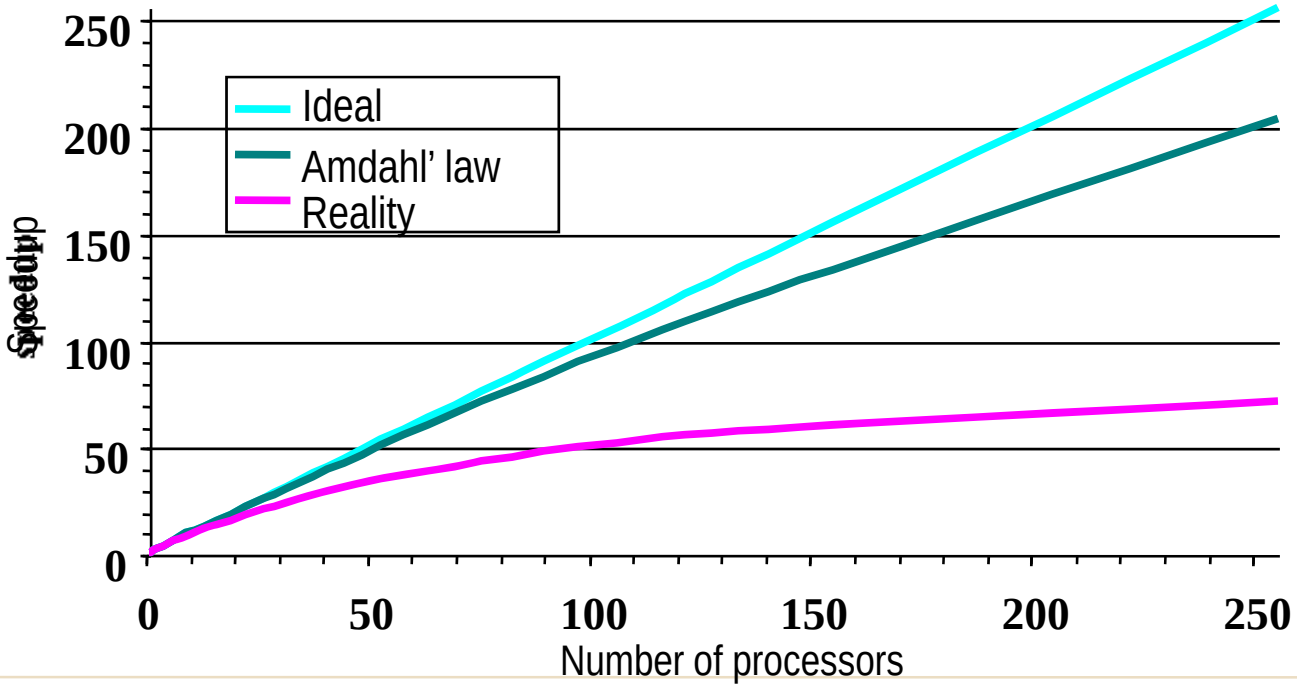
Limitations of Parallelization

- Considerations
  - Is the problem parallelizable at all?
  - Time to re-write code
- Theoretical Upper Limits
  - Amdahl's Law
- Practical Limits
  - Load balancing
  - Non-computational sections
  - Memory Access

Limitations of Parallelization

Informally, ***Amdahl's Law*** states:

Speedup is the ratio of the time required to run a code on one processor to the time required to run the same code on multiple (N) processors



Limitations of Parallelization

- Amdahl's Law limits scalability by many things:
  - Communications
  - I/O
  - Load balancing (waiting)
  - Scheduling (shared processors or memory)

Thank You

Questions or Comments?