Code Analysis Tool Parallelisation and Grid Computing

Anna-Jayne Metcalfe
Riverblade Limited
http://www.riverblade.co.uk

Riverblade

Effective Code Analysis

- For a code analysis tool to be effective, it needs to:
  - Be straightforward to configure and use
  - Produce meaningful and accurate results
  - Produce analysis results reasonably quickly
  - Be accepted by those who need it
- In this session we are going to focus on analysis speed and techniques for improving it.

Riverblade

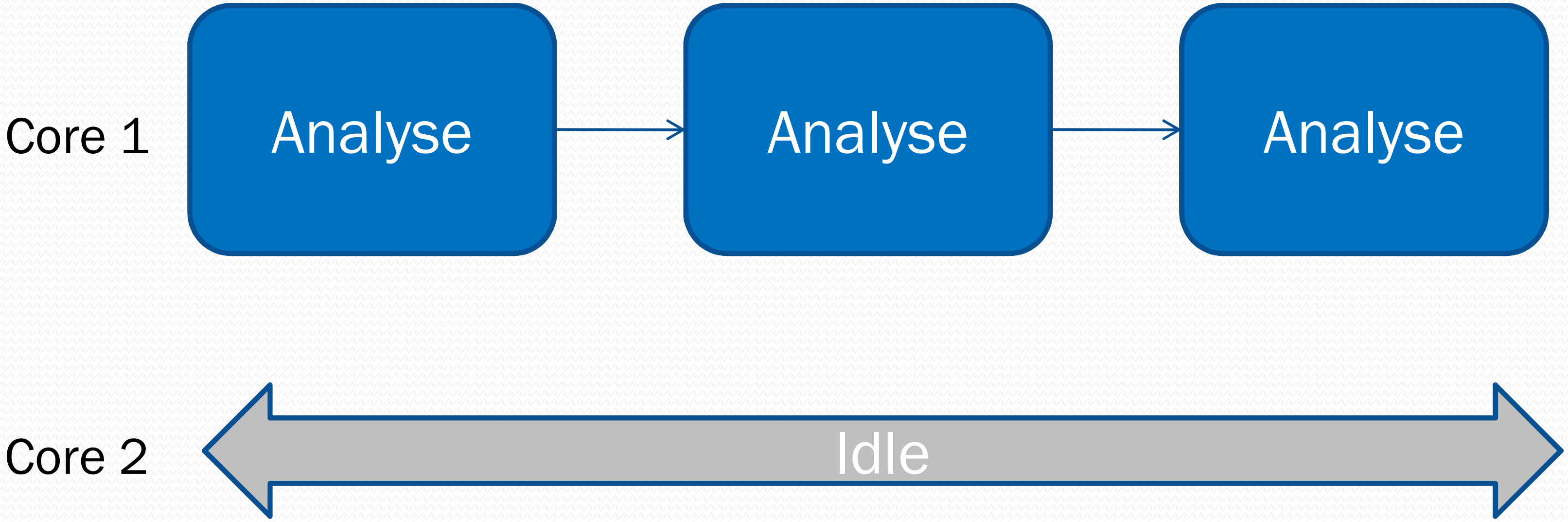
The Problem

- Detailed source code analysis is inherently slow
- C++ in particular is complex and extremely hard to analyse correctly
- Include files are **very** inefficient from an analysis perspective
  - Repeated opening of common include files, each of which must be preprocessed etc.

Riverblade

Conventional Code Analysis

- Simple, but inefficient:



Core 1

Analyse

Analyse

Analyse

Core 2

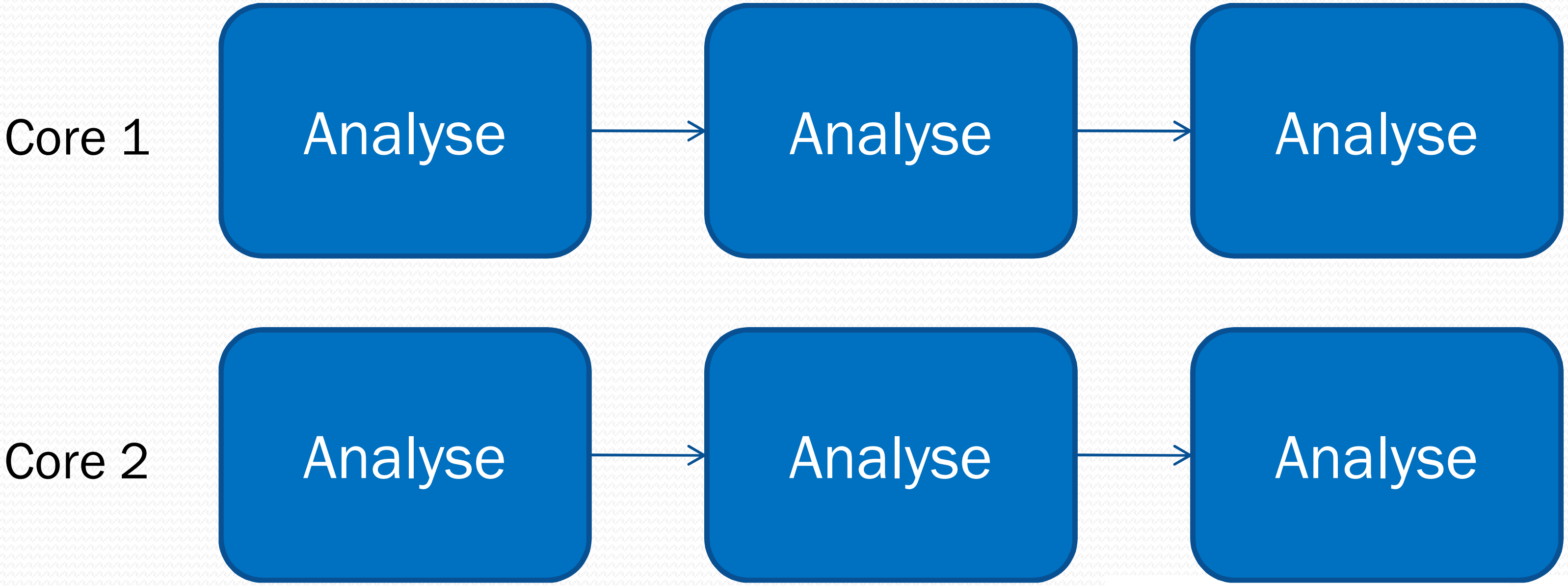
Idle

e.g. 4 hrs 26 mins for an example 178kLOC codebase (on a 2.2GHz Opteron with 5GB RAM)

Riverblade

Local Parallel Analysis

- Independent analysis tasks are **very** amenable to parallelisation:



Core 1

Analyse → Analyse → Analyse

Core 2

Analyse → Analyse → Analyse

e.g. 2 hrs 22 mins for a 178kLOC codebase (2.2GHz Opteron with 5GB RAM). 1.9x faster...

Riverblade

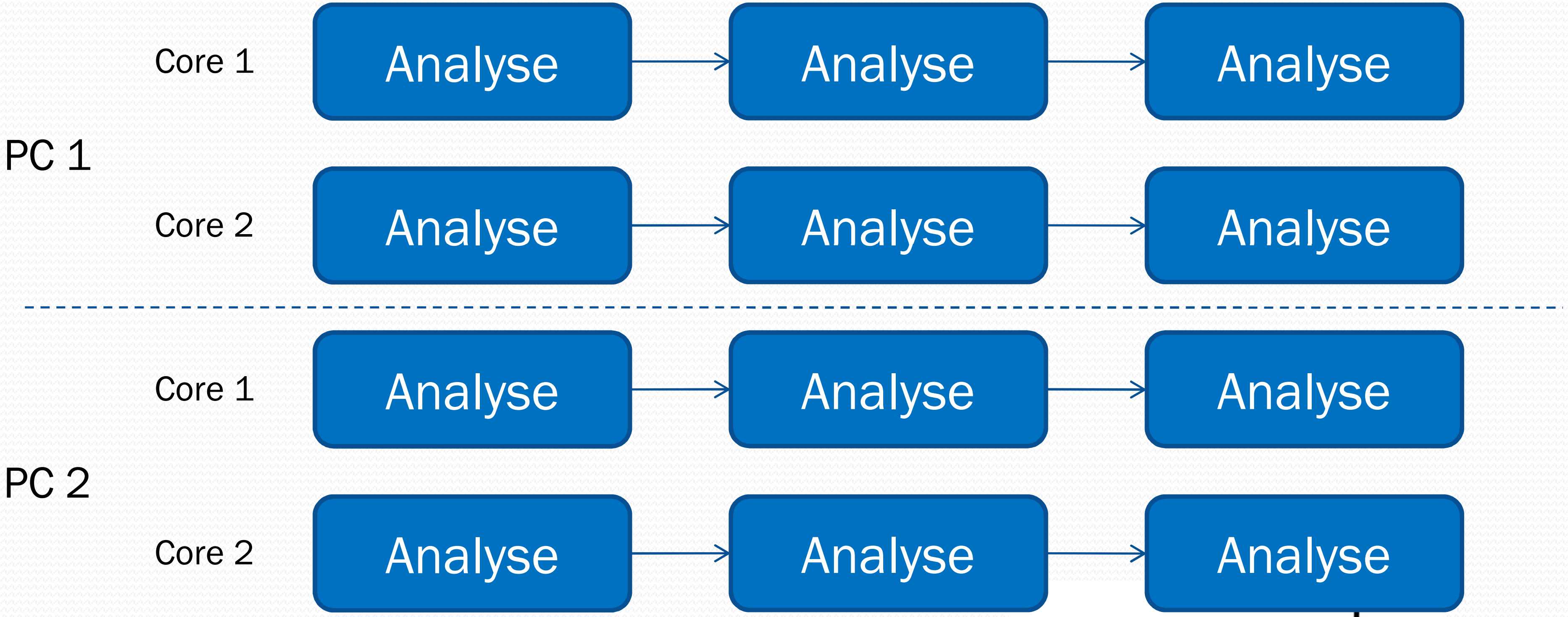
Parallel Analysis Considerations

- Parallelisation works well if tasks are independent
- Local parallelisation is quite straightforward:
  - Generate the command line
  - Run the task in a thread pool
  - Collate results as each task completes
- Optimal number of parallel tasks is rather subjective
  - 3 seems to work well for a dual core machine

Riverblade

Grid Parallel Analysis

- Independent analysis tasks are amenable to parallelisation:



PC 1

Core 1: Analyse → Analyse → Analyse

Core 2: Analyse → Analyse → Analyse

PC 2

Core 1: Analyse → Analyse → Analyse

Core 2: Analyse → Analyse → Analyse

e.g. 15 mins 10 secs for a 178kLOC codebase
(16 cores; aggregate CPU speed 40GHz). 17x faster!

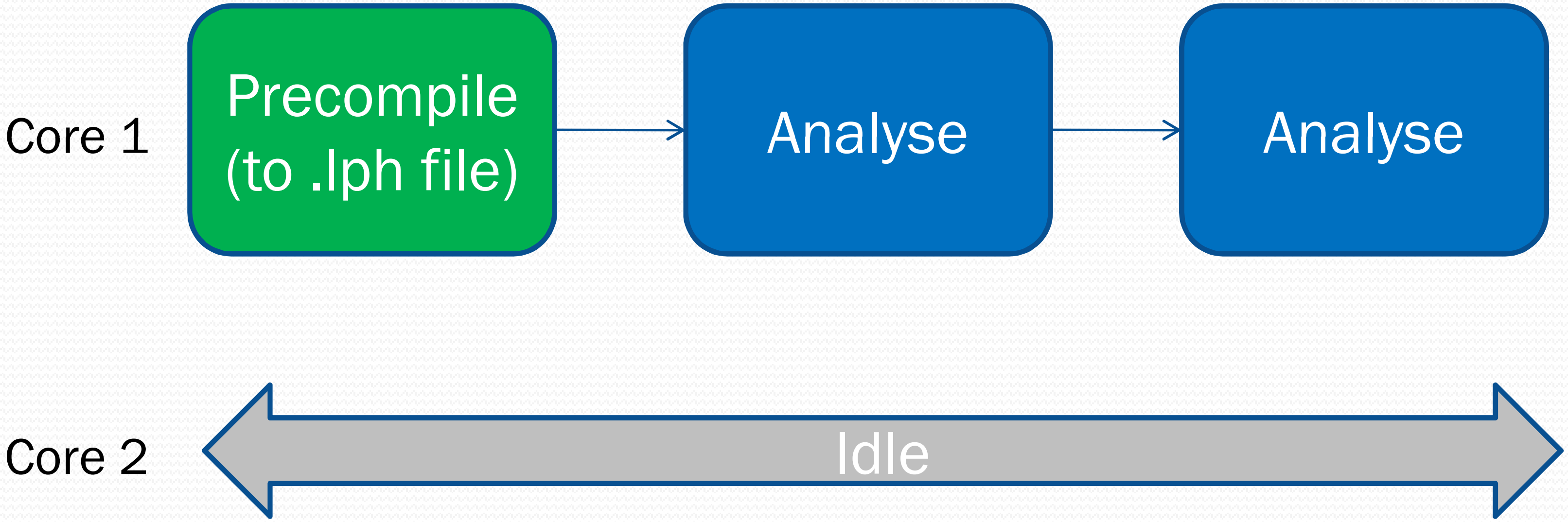
Riverblade

Grid Analysis Considerations

- Significantly trickier than local parallelisation
  - Include files and preprocessing can be a problem
  - Efficient use of network bandwidth is important
- Two contrasting approaches:
  - Preprocessing on the local system (distcc etc.)
  - File system virtualisation and caching (Xoreax XGE and possibly Electric Accelerator)
- With a virtualised grid, a networked solution is not much more difficult than a local parallelised solution

PCH Code Analysis

- PC-lint 9 can use “precompiled header file” techniques to speed up the analysis:

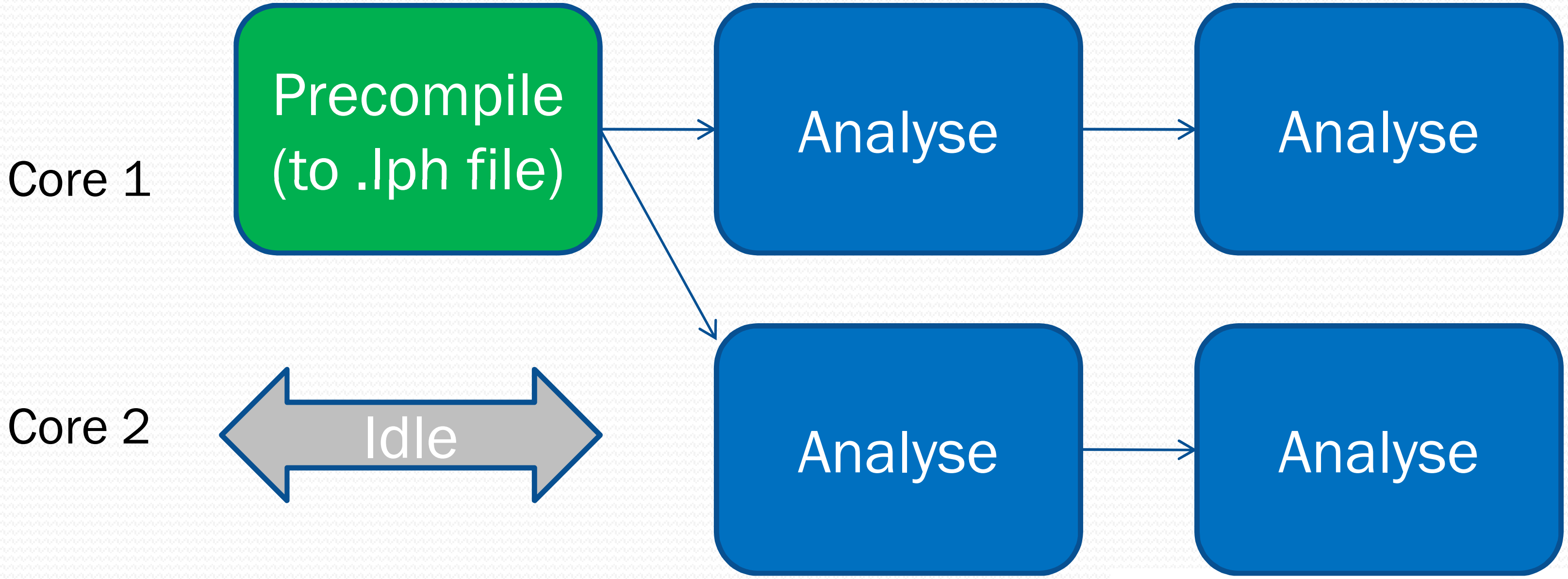


e.g. 1 hr 13 mins for a 178kLOC codebase (2.2GHz Opteron with 5GB RAM). 3.6x faster…

Riverblade

Parallel Analysis with PCH

- Requires reasonably careful scheduling:

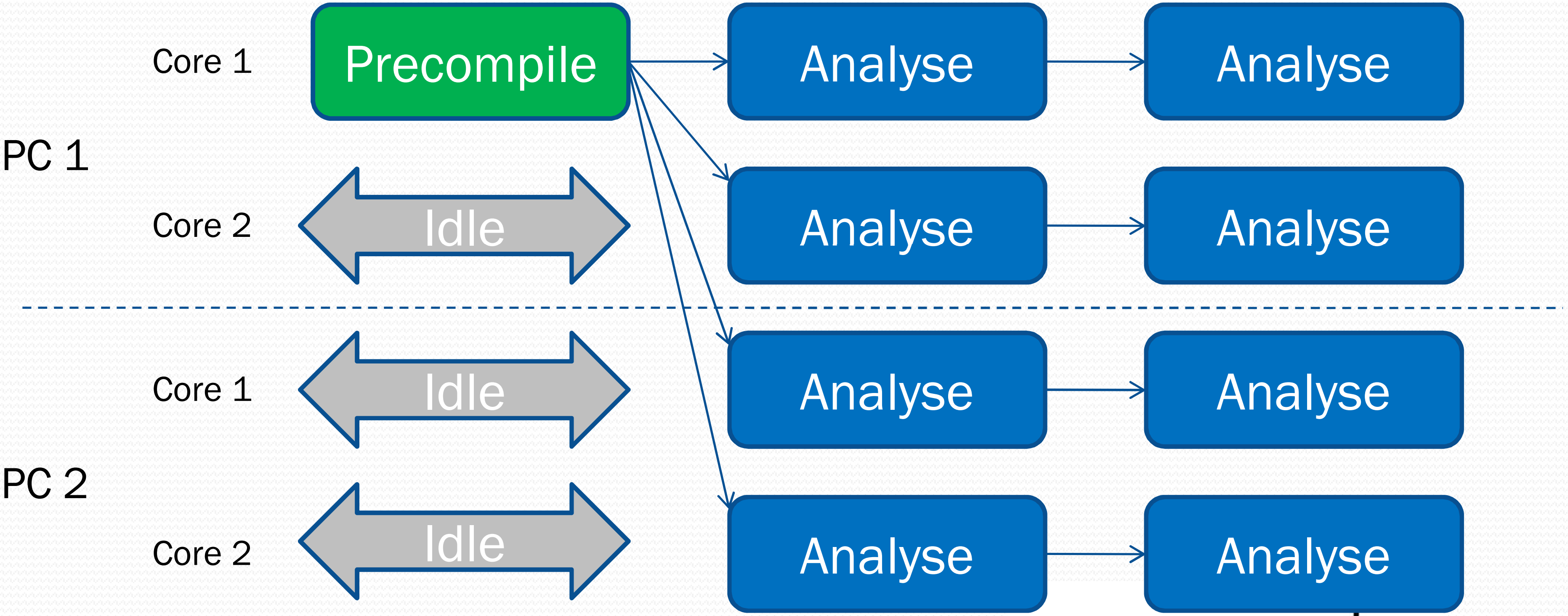


e.g. 46 mins 5 secs for a 178kLOC codebase (2.2GHz Opteron with 5GB RAM). 5.8x faster…

Riverblade

Grid Analysis with PCH

- Requires **very** careful scheduling:



May or may not be faster than a non-PCH grid solution, depending on the scheduling!

Conclusion

- Comparative Timings:

Description	Time	Speed
Single threaded; No PCH	4:25:44	x1.0
Parallelised (3 threads)	2:22:16	x1.9
Single threaded with PCH	1:13:41	x3.6
Parallelised with PCH	0:46:05	x5.8
Grid (16 cores; 40GHZ)	0:15:10	x17.5
Grid (23 cores; 54.7GHz)	0:12:10	x21.5
Grid with PCH	???	???

- Any Questions?