Characterizing Imbalance in Large-Scale Parallel Programs

David Böhme
September 26, 2013
Need for Performance Analysis Tools

- Amount of parallelism in Supercomputers keeps growing
  - Efficient resource usage depends on software performance

Jugene (2008): 65,536 cores

Juqueen (2013): 458,752 cores
Imbalance limits parallel efficiency

Imbalance leads to wait states at synchronization points

Late-sender wait state

Goal: Locate inefficient parallelism and quantify its impact
Two novel analysis methods

Root-cause analysis

Critical-path analysis

Determine delays that cause wait states.
Two novel analysis methods

Root-cause analysis

Critical-path analysis

Identify activities that determine program runtime.
Outline

Parallel Performance Analysis

Root-Cause Analysis

  Concepts
  Case study

Critical-Path Analysis

  The critical path
  Critical-path imbalance indicator
  Analysis of MPMD programs

Implementation

  Parallel trace replay
  Scalability evaluation
Performance analysis tools

- Tools help understand performance, but
  - Profiling provides limited insights
  - Tracing produces too much data to analyze manually

Time profile display in TAU

Event-trace timeline visualization in Vampir
Imbalance analysis pitfalls

- Profilers underestimate impact of imbalance
  - Data aggregation hides dynamic imbalance effects
Imbalance analysis pitfalls

- Profilers underestimate impact of imbalance
  - Data aggregation hides dynamic imbalance effects

- Analysis solution needs to retain performance dynamics
Imbalance analysis pitfalls

- Profilers underestimate impact of imbalance
  - Data aggregation hides dynamic imbalance effects

- Analysis solution needs to retain performance dynamics
- Use automatic trace analysis
Analysis workflow

Source modules

Instrumenter
compiler / linker

Instrumented executable

Application run

Local event traces

Parallel analysis

Global analysis

report

Which problem?

Where in the code?

Which process?
Analysis workflow

- Source modules
- Instrumenter
- Compiler / Linker
- Instrumented executable
- Application run
- Local event traces
- Parallel analysis
- Global analysis
- Report

Which problem?
Where in the code?
Which process?
Analysis workflow

Source modules

Instrumenter compiler / linker

Instrumented executable

Application run

Local event traces

Parallel analysis

Global analysis report

Which problem?

Where in the code?

Which process?
Analysis workflow

1. Source modules
2. Instrumenter compiler / linker
3. Instrumented executable
4. Application run
5. Local event traces
6. Parallel analysis
7. Global analysis report

Which problem?
Where in the code?
Which process?
Parallel Performance Analysis

Root-Cause Analysis
  Concepts
  Case study

Critical-Path Analysis
  The critical path
  Critical-path imbalance indicator
  Analysis of MPMD programs

Implementation
  Parallel trace replay
  Scalability evaluation
Delay as root cause of wait states

A delay is some additional activity on one process that causes a wait state at a synchronization point.

Delay on process 1 causes a late-sender wait state on process 2.
Wait-state propagation

- Wait states can propagate

![Diagram showing wait-state propagation](image-url)
Wait-state propagation

- Wait states can propagate
Wait-state propagation

- Wait states can propagate
- Account for propagation in analysis
  - Extended wait-state classification
  - Incorporate long-distance effects in calculation of delay costs
Wait-state classification

Distinguish propagating and terminal wait states
Wait-state classification

Distinguish propagating and terminal wait states
Wait-state classification

Distinguish **propagating** and **terminal** wait states
Assigning delay costs

- **Delay costs** represent amount of wait time caused by a delay
  - **Short-term costs** represent wait states caused directly
  - **Long-term costs** represent wait states caused via propagation
Case study: CESM sea ice model

- Analysis of imbalance in CESM sea ice model
- Performance data mapped onto application topology

CICE setup: 2048 processes on BG/P, $1^\circ$ dipole grid, cartesian grid decomposition
Case study: CESM sea ice model

- Analysis of imbalance in CESM sea ice model
- Performance data mapped onto application topology

CICE setup: 2048 processes on BG/P, 1° dipole grid, cartesian grid decomposition
CESM sea ice model: wait-state formation

Distribution of delay costs

25% Short-term
75% Long-term

Propagating wait states

Terminal wait states
Parallel Performance Analysis

Root-Cause Analysis
  Concepts
  Case study

Critical-Path Analysis
  The critical path
  Critical-path imbalance indicator
  Analysis of MPMD programs

Implementation
  Parallel trace replay
  Scalability evaluation
Critical-path analysis

- Use automatic trace analysis to extract the critical path

Critical path in a parallel program (shown in red)
Critical-path analysis

- Use automatic trace analysis to extract the critical path
- Performance indicators show bottlenecks at single glance

Critical path in a parallel program (shown in red)
Critical-path profile and imbalance

Critical-path profile shows wall-clock time consumption.

Imbalance indicator finds inefficient parallelism.

Imbalance = \( T_{\text{critical}} - T_{\text{avg}} \)

Timeline
Critical-path profile and imbalance

Critical-path profile shows wall-clock time consumption

Summary profile

Timeline

Imbalance = \( T_{\text{critical}} - T_{\text{avg}} \)
Critical-path profile and imbalance

- **Critical-path profile** shows wall-clock time consumption
Critical-path profile and imbalance

- **Critical-path profile** shows wall-clock time consumption
- **Critical imbalance indicator** finds inefficient parallelism
  - Imbalance $= T_{critical} - T_{avg}$
Example: PEPC

- Analysis of plasma-physics code PEPC using 512 processes on Blue Gene/P
Example: PEPC

- Analysis of plasma-physics code PEPC using 512 processes on Blue Gene/P
- Profile metrics underestimate performance impact of `tree_walk` kernel due to dynamic load imbalance
Analysis of MPMD programs

- Processes execute different activities
  - E.g. master-worker

Heterogeneous decomposition in ddcMDD.
Image from Richards et al.: Beyond Homogeneous Decomposition, SC’10
Analysis of MPMD programs

- Processes execute different activities
  - E.g. master-worker
- Complex imbalance analysis issues
  - Not supported by existing tools
  - Imbalance quantification needs to incorporate partition sizes
  - More knobs to tune

Heterogeneous decomposition in ddcMDD.
Image from Richards et al.: Beyond Homogeneous Decomposition, SC’10
Performance impact indicators

- Denote allocation-time costs of imbalance
  - Map wait time onto critical-path activities with excess time
  - Distinguish *intra-partition* and *inter-partition* imbalance costs

High intra-partition costs, low inter-partition costs

Very high inter-partition costs, low intra-partition costs
Example: ddcMD

ddcMD molecular dynamics simulation on Blue Gene/P

- *Particle* and *mesh* forces calculated in different partitions

D. Richards et al.: Beyond Homogeneous Decomposition, SC’10
Example: ddcMD

ddcMD molecular dynamics simulation on Blue Gene/P

- Particle and mesh forces calculated in different partitions
- Fixed partition sizes: 3840+256 processes
Example: ddcMD

ddcMD molecular dynamics simulation on Blue Gene/P

- *Particle* and *mesh* forces calculated in different partitions
- Fixed partition sizes: 3840+256 processes
- Tune mesh size to adjust load balance

D. Richards et al.: Beyond Homogeneous Decomposition, SC’10
ddcMD: mesh size tuning

- Small mesh size increases workload of particle tasks
- Increasing mesh size shifts critical path to mesh tasks
Parallel Performance Analysis

Root-Cause Analysis
  Concepts
  Case study

Critical-Path Analysis
  The critical path
  Critical-path imbalance indicator
  Analysis of MPMD programs

Implementation
  Parallel trace replay
  Scalability evaluation
Implementation

- Integrated in Scalasca trace analysis toolset

- Highly scalable parallel trace analysis
  - So far only for wait-state detection
Parallel trace replay

- Application records timestamped communication events
  - One trace file per process
Parallel trace replay

- Application records timestamped communication events
  - One trace file per process
- Analysis processes traverse traces in parallel
  - Exchange information at original synchronization points
Trace analysis extensions

- Use multiple replay passes
- Backward replay lets data travel from effect to cause
Delay detection via backward replay

1. Identify synchronization interval
2. Determine time vectors \( \vec{t}_s \) and \( \vec{t}_r \)
3. Transfer time vector \( \vec{t}_r \)
4. Locate delay
5. Calculate costs

Slide 28    September 26, 2013    David Böhme
Delay detection via backward replay

1. Identify synchronization interval

2. Determine time vectors $\vec{t}_{s}$ and $\vec{t}_{r}$

3. Transfer time vector $\vec{t}_{r}$

4. Locate delay

5. Calculate costs
Delay detection via backward replay

1. Identify synchronization interval
2. Determine time vectors $\vec{t}_s$ and $\vec{t}_r$

Diagram:

- Processes 1 and 2
- Time vectors $\vec{t}_s$ and $\vec{t}_r$
- Synchronization interval
- Wait state
Delay detection via backward replay

1. Identify synchronization interval
2. Determine time vectors $\vec{t_s}$ and $\vec{t_r}$
3. Transfer time vector $\vec{t_r}$

Diagram:
- Processes 1 and 2
- Synchronization interval
- Wait state
- Time vectors $\vec{t_s}$ and $\vec{t_r}$
Delay detection via backward replay

1. Identify synchronization interval
2. Determine time vectors $\vec{t}_s$ and $\vec{t}_r$
3. Transfer time vector $\vec{t}_r$
4. Locate delay
Delay detection via backward replay

1. Identify synchronization interval
2. Determine time vectors $\vec{t}_s$ and $\vec{t}_r$
3. Transfer time vector $\vec{t}_r$
4. Locate delay
5. Calculate costs
Critical-path extraction in backward replay
Critical-path extraction in backward replay

1. c=false
2. c=true
3. c=false

Processes:
- Process 1
- Process 2
- Process 3

Time line with processes and critical path markings.
Critical-path extraction in backward replay

1
2
3
R
R
R

c=true

R

R

R

R

c=false

R

R

R

R

c=false

c=true

c=true

c=false
Critical-path extraction in backward replay

Slide 29  September 26, 2013  David Böhme
Scalability

Scalability of root-cause and critical-path analysis for the Sweep3D benchmark on Blue Gene/P
Scalability of root-cause and critical-path analysis for the Sweep3D benchmark on Blue Gene/P
Scalability of root-cause and critical-path analysis for the Sweep3D benchmark on Blue Gene/P
Summary

Two novel methods to locate and quantify imbalance

Root-cause analysis

Identifies delays and explains formation of wait states
Summary

Two novel methods to locate and quantify imbalance

Root-cause analysis

Identifies delays and explains formation of wait states

Critical-path analysis

Determines impact of inefficiency on runtime
Summary

Two novel methods to locate and quantify imbalance

Root-cause analysis
Identifies delays and explains formation of wait states

Critical-path analysis
Determines impact of inefficiency on runtime

Highly scalable implementation for $O(100k)$ processes
Thank you

Further reading:

SPECMPi case studies

Tracing run-time overhead and trace replay times

Critical-path vs. profile load imbalance metrics