Performance Engineering

Parallel Performance

Elisabet Molin, Xavi Aguilar

elimo@pdc.kth.se

PDC

KTH, Sweden

PDC Center for
High Performance Computing

Why Parallel Performance?

- Reduce calculation time

PDC Center for
High Performance Computing

Why Parallel Performance?

- Reduce calculation time
- Expand to solve new problems

PDC Center for
High Performance Computing

Why Parallel Performance?

- Reduce calculation time
- Expand to solve new problems
- Choosing appropriate computers

PDC Center for
High Performance Computing

Characteristics of a code

- Communication pattern

PDC Center for
High Performance Computing

Characteristics of a code

- Communication pattern
- Load balance

PDC Center for
High Performance Computing

Characteristics of a code

- Communication pattern
- Load balance
- Number of individual computations

PDC Center for
High Performance Computing

Characteristics of a code

- Communication pattern
- Load balance
- Number of individual computations
- Memory usage

PDC Center for
High Performance Computing

Characteristics of a code

- Communication pattern
- Load balance
- Number of individual computations
- Memory usage
- Data I/O pattern

PDC Center for
High Performance Computing

Characteristics of a code

- Communication pattern
- Load balance
- Number of individual computations
- Memory usage
- Data I/O pattern
- Size and layout of data sets

PDC Center for
High Performance Computing

Know your enemy

What is expensive and slow?

- Data transfer

PDC Center for
High Performance Computing

Know your enemy

What is expensive and slow?

- Data transfer
- File I/O

PDC Center for
High Performance Computing

Know your enemy

What is expensive and slow?

- Data transfer
- File I/O
- Bad memory utilization

PDC Center for
High Performance Computing

Know your enemy

What is expensive and slow?

- Data transfer
- File I/O
- Bad memory utilization
- Serial code sections (Amdahl's law)

PDC Center for
High Performance Computing

Where to optimize

- Premature optimization is the root of all evil

PDC Center for
High Performance Computing

Where to optimize

- Premature optimization is the root of all evil
- 90 % of the time will usually be spent on 10 % of the code

PDC Center for
High Performance Computing

Where to optimize

- Premature optimization is the root of all evil
- 90 % of the time will usually be spent on 10 % of the code
- Won’t reach theoretical peak performance

PDC Center for
High Performance Computing

Presenting Performance data

To be able to understand a graph the following is required:

- What input data was used? (dense/sparse, size, precision. . .)

PDC Center for
High Performance Computing

Presenting Performance data

To be able to understand a graph the following is required:

- What input data was used? (dense/sparse, size, precision...)
- What computer was used? (memory, cpu, interconnect...)

Presenting Performance data

To be able to understand a graph the following is required:

- What input data was used? (dense/sparse, size, precision. . .)
- What computer was used? (memory, cpu, interconnect. . .)
- How many nodes were used?

Presenting Performance data

To be able to understand a graph the following is required:

- What input data was used? (dense/sparse, size, precision. . .)
- What computer was used? (memory, cpu, interconnect. . .)
- How many nodes were used?
- How many runs were averaged? (error margins)

PDC Center for
High Performance Computing

Presenting Performance data

To be able to understand a graph the following is required:

- What input data was used? (dense/sparse, size, precision. . .)
- What computer was used? (memory, cpu, interconnect. . .)
- How many nodes were used?
- How many runs were averaged? (error margins)
- What is the base line? (what is the comparison made against)

PDC Center for High Performance Computing

Typical graphs

- T_1^S shortest time for *the best serial program*.

PDC Center for
High Performance Computing

Typical graphs

- T_1^S shortest time for *the best serial program*.
- T_1 shortest time with the multicomputer program on one node

PDC Center for
High Performance Computing

Typical graphs

- T_1^S shortest time for *the best serial program*.
- T_1 shortest time with the multicomputer program on one node
- T_p: execution time for $\mathtt{p}$-node computation

PDC Center for
High Performance Computing

Typical graphs

- T_1^S shortest time for *the best serial program*.
- T_1 shortest time with the multicomputer program on one node
- T_p: execution time for $\mathtt{p}$-node computation
- Speed-up

KTH VETENSKAP OCH KONST

PDC Center for
High Performance Computing

Typical graphs

- T_1^S shortest time for *the best serial program*.
- T_1 shortest time with the multicomputer program on one node
- T_p: execution time for `p`-node computation
- Speed-up

  Absolute $S_p = \frac{T_1^S}{T_p}$

PDC Center for
High Performance Computing

Typical graphs

- T_1^S shortest time for *the best serial program*.
- T_1 shortest time with the multicomputer program on one node
- T_p: execution time for $\texttt{p}$-node computation
- Speed-up

  Absolute $S_p = \frac{T_1^S}{T_p}$

  Relative $S_p^{rel} = \frac{T_1}{T_p}$

PDC Center for
High Performance Computing

Typical graphs

- T_1^S shortest time for *the best serial program*.
- T_1 shortest time with the multicomputer program on one node
- T_p: execution time for `p`-node computation
- Speed-up

  Absolute $S_p = \frac{T_1^S}{T_p}$

  Relative $S_p^{rel} = \frac{T_1}{T_p}$

  Absolute speed-up is *improvement achieved by parallelisation*

KTH VETENSKAP OCH KONST

PDC Center for High Performance Computing

Typical graphs

- T_1^S shortest time for *the best serial program*.
- T_1 shortest time with the multicomputer program on one node
- T_p: execution time for `p`-node computation
- Speed-up

  Absolute $S_p = \frac{T_1^S}{T_p}$

  Relative $S_p^{rel} = \frac{T_1}{T_p}$

  Absolute speed-up is *improvement achieved by parallelisation*
- Efficiency

PDC Center for
High Performance Computing

Typical graphs

- T_1^S shortest time for *the best serial program*.
- T_1 shortest time with the multicomputer program on one node
- T_p: execution time for $\mathtt{p}$-node computation
- Speed-up

  Absolute $S_p = \frac{T_1^S}{T_p}$

  Relative $S_p^{rel} = \frac{T_1}{T_p}$

  Absolute speed-up is *improvement achieved by parallelisation*
- Efficiency

  Absolute $\eta_p = \frac{S_p}{p}$

PDC Center for
High Performance Computing

Typical graphs

- T_1^S shortest time for *the best serial program*.
- T_1 shortest time with the multicomputer program on one node
- T_p: execution time for $\mathtt{p}$-node computation
- Speed-up

  Absolute $S_p = \frac{T_1^S}{T_p}$

  Relative $S_p^{rel} = \frac{T_1}{T_p}$

  Absolute speed-up is *improvement achieved by parallelisation*
- Efficiency

  Absolute $\eta_p = \frac{S_p}{p}$

  Relative $\eta_p^{rel} = \frac{S_p^{rel}}{p}$

PDC Center for High Performance Computing

Typical graphs

- T_1^S shortest time for *the best serial program*.
- T_1 shortest time with the multicomputer program on one node
- T_p: execution time for $\mathtt{p}$-node computation
- Speed-up

  Absolute $S_p = \frac{T_1^S}{T_p}$

  Relative $S_p^{rel} = \frac{T_1}{T_p}$

  Absolute speed-up is *improvement achieved by parallelisation*
- Efficiency

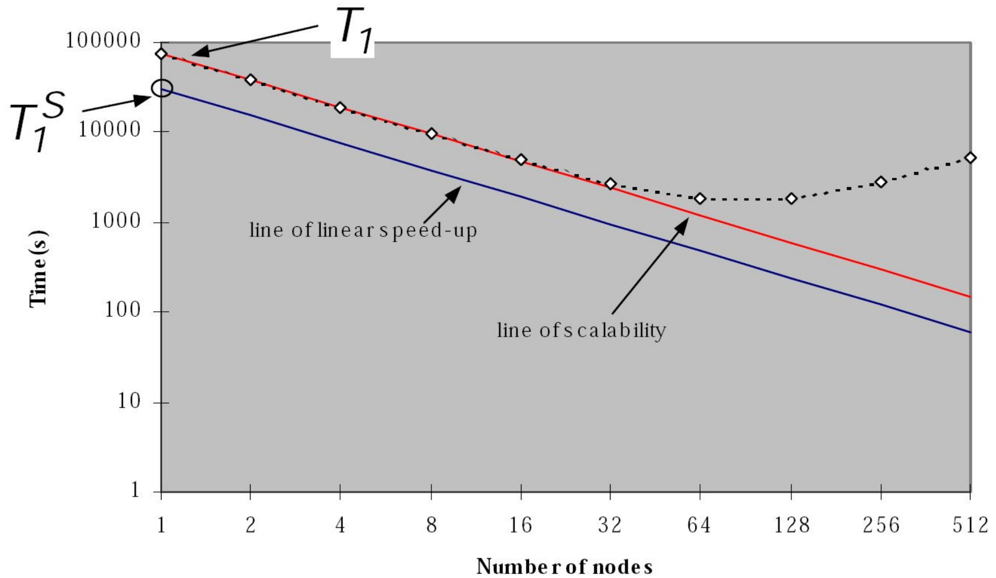
  Absolute $\eta_p = \frac{S_p}{p}$

  Relative $\eta_p^{rel} = \frac{S_p^{rel}}{p}$

  Relative efficiency is a measure of *scalability*

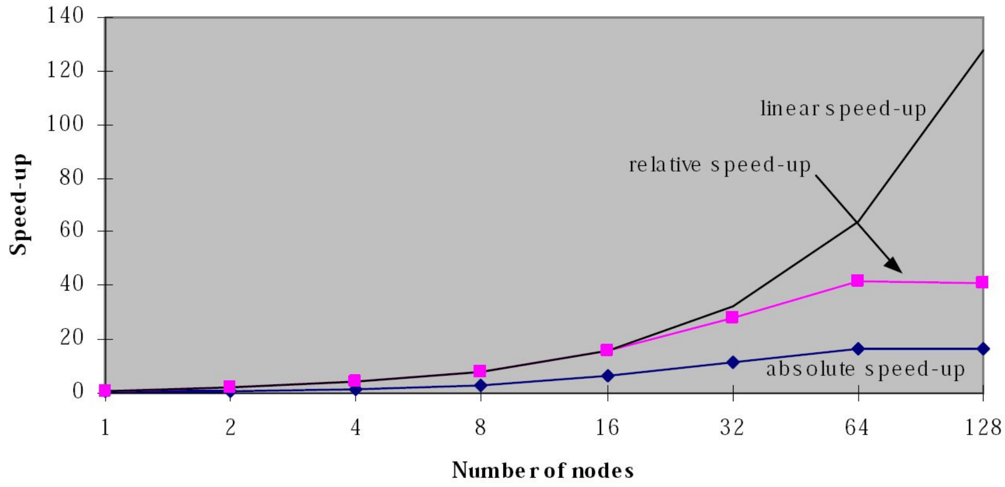
PDC Center for
High Performance Computing

Execution time



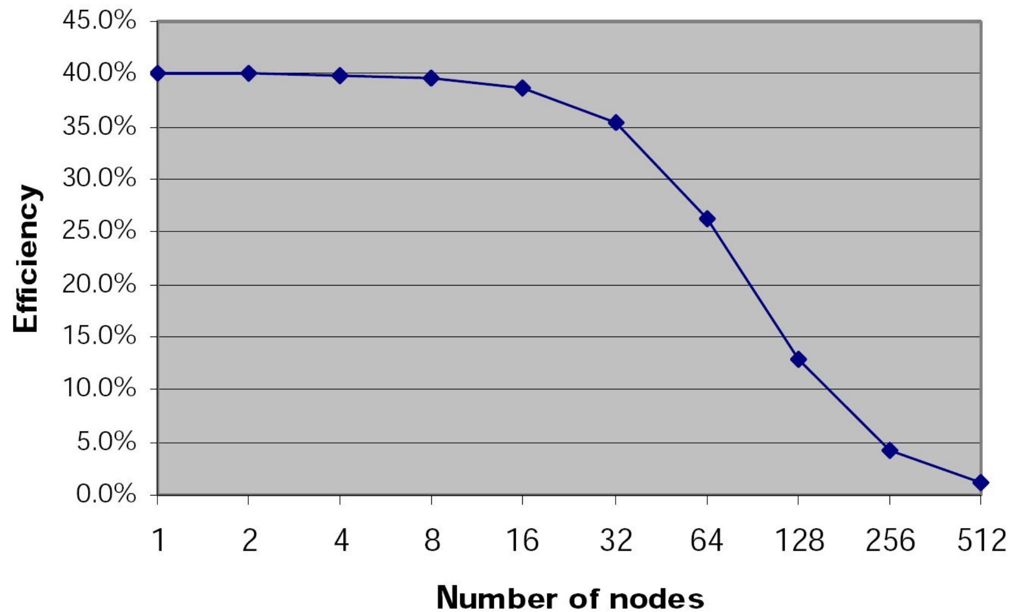
PDC Center for
High Performance Computing

Speed-up



Absolute $S_p = \frac{T_1^S}{T_p}$, relative $S_p^{rel} = \frac{T_1}{T_p}$

Efficiency



$$\eta_p^{rel} = \frac{S_p^{rel}}{p} = \frac{\frac{T_1}{T_p}}{p}$$

PDC Center for
High Performance Computing

Where to start

Performance improvement doesn't always require changing your code.

- Compiler optimization flags

PDC Center for
High Performance Computing

Where to start

Performance improvement doesn't always require changing your code.

- Compiler optimization flags
- Libraries (LAPACK/BLAS, FFTW. . .)

PDC Center for
High Performance Computing

Where to start

Performance improvement doesn't always require changing your code.

- Compiler optimization flags
- Libraries (LAPACK/BLAS, FFTW. . .)
- Parallel libraries (ScaLAPACK, FFTW. . .)

PDC Center for
High Performance Computing

Where to start

Performance improvement doesn't always require changing your code.

- Compiler optimization flags
- Libraries (LAPACK/BLAS, FFTW. . .)
- Parallel libraries (ScaLAPACK, FFTW. . .)
- Use the precision you actually need (float vs. double)

PDC Center for
High Performance Computing

Where to start

Performance improvement doesn't always require changing your code.

- Compiler optimization flags
- Libraries (LAPACK/BLAS, FFTW...)
- Parallel libraries (ScaLAPACK, FFTW...)
- Use the precision you actually need (float vs. double)
- Get to know the computer architecture

PDC Center for
High Performance Computing

Where to start

Performance improvement doesn't always require changing your code.

- Compiler optimization flags
- Libraries (LAPACK/BLAS, FFTW. . .)
- Parallel libraries (ScaLAPACK, FFTW. . .)
- Use the precision you actually need (float vs. double)
- Get to know the computer architecture
- Communicate according to network topology

PDC Center for
High Performance Computing

Where to start

Performance improvement doesn't always require changing your code.

- Compiler optimization flags
- Libraries (LAPACK/BLAS, FFTW...)
- Parallel libraries (ScaLAPACK, FFTW...)
- Use the precision you actually need (float vs. double)
- Get to know the computer architecture
- Communicate according to network topology
- Place data according to network topology

PDC Center for
High Performance Computing

Measuring Performance

A. External timers

PDC Center for
High Performance Computing

Measuring Performance

A. External timers

B. Internal time

PDC Center for
High Performance Computing

Measuring Performance

A. External timers

B. Internal time

C. Performance counters

PDC Center for
High Performance Computing

Measuring Performance

A. External timers

B. Internal time

C. Performance counters

D. Profilers

PDC Center for
High Performance Computing

Measuring Performance

A. External timers

B. Internal time

C. Performance counters

D. Profilers

E. Call Tracing

PDC Center for
High Performance Computing

A. External timers

- Measuring wall clock time on executable

PDC Center for
High Performance Computing

A. External timers

- Measuring wall clock time on executable
- `/usr/bin/time`

PDC Center for
High Performance Computing

A. External timers

- Measuring wall clock time on executable
- `/usr/bin/time`
- `Real`: Time from beginning till end

PDC Center for
High Performance Computing

A. External timers

- Measuring wall clock time on executable
- `/usr/bin/time`
- `Real`: Time from beginning till end
- `User`: CPU time spent in user code

PDC Center for
High Performance Computing

A. External timers

- Measuring wall clock time on executable
- `/usr/bin/time`
- `Real`: Time from beginning till end
- `User`: CPU time spent in user code
- `Sys`: CPU time spent in system code

PDC Center for
High Performance Computing

A. External timers

- Measuring wall clock time on executable
- `/usr/bin/time`
- `Real`: Time from beginning till end
- `User`: CPU time spent in user code
- `Sys`: CPU time spent in system code

\+ Easy to use

PDC Center for
High Performance Computing

A. External timers

- Measuring wall clock time on executable
- `/usr/bin/time`
- `Real`: Time from beginning till end
- `User`: CPU time spent in user code
- `Sys`: CPU time spent in system code

+ Easy to use

- Execution time $\geq$ CPU time

PDC Center for
High Performance Computing

A. External timers

- Measuring wall clock time on executable
- `/usr/bin/time`
- `Real`: Time from beginning till end
- `User`: CPU time spent in user code
- `Sys`: CPU time spent in system code

+ Easy to use

- Execution time $\geq$ CPU time
- Different definition on different systems

PDC Center for High Performance Computing

A. External timers

- Measuring wall clock time on executable
- `/usr/bin/time`
- `Real`: Time from beginning till end
- `User`: CPU time spent in user code
- `Sys`: CPU time spent in system code

+ Easy to use

- Execution time $\geq$ CPU time
- Different definition on different systems
- Depend on the load of the system, OS interference, etc

PDC Center for
High Performance Computing

A. External timers

- Measuring wall clock time on executable
- `/usr/bin/time`
- `Real`: Time from beginning till end
- `User`: CPU time spent in user code
- `Sys`: CPU time spent in system code

+ Easy to use

- Execution time $\geq$ CPU time
- Different definition on different systems
- Depend on the load of the system, OS interference, etc

! Multithreaded execution (on one node)

$$T = t_i^{last} - t_0^{first}$$

t_0^{first} — first thread starts execution t_i^{last} — last thread finishes.

PDC Center for High Performance Computing

B. Internal Timers

- Source code adapted to start, stop and save timers

PDC Center for
High Performance Computing

B. Internal Timers

- Source code adapted to start, stop and save timers
- C calls:

PDC Center for
High Performance Computing

B. Internal Timers

- Source code adapted to start, stop and save timers
- C calls:

  `gettimeofday()`, `time()` — time since January 1, 1970

PDC Center for
High Performance Computing

B. Internal Timers

- Source code adapted to start, stop and save timers
- C calls:

  `gettimeofday()`, `time()` — time since January 1, 1970

  `clock()`, approximation of processor time

PDC Center for
High Performance Computing

B. Internal Timers

- Source code adapted to start, stop and save timers
- C calls:

  `gettimeofday()`, `time()` — time since January 1, 1970

  `clock()`, approximation of processor time

  `MPI_Wtime()`, for MPI codes

PDC Center for
High Performance Computing

B. Internal Timers

- Source code adapted to start, stop and save timers
- C calls:

  `gettimeofday()`, `time()` — time since January 1, 1970

  `clock()`, approximation of processor time

  `MPI_Wtime()`, for MPI codes
- Fortran calls:

PDC Center for
High Performance Computing

B. Internal Timers

- Source code adapted to start, stop and save timers
- C calls:

  `gettimeofday()`, `time()` — time since January 1, 1970

  `clock()`, approximation of processor time

  `MPI_Wtime()`, for MPI codes

- Fortran calls:

  `system_clock()`, wall clock time

PDC Center for
High Performance Computing

B. Internal Timers

- Source code adapted to start, stop and save timers
- C calls:

  `gettimeofday()`, `time()` — time since January 1, 1970

  `clock()`, approximation of processor time

  `MPI_Wtime()`, for MPI codes

- Fortran calls:

  `system_clock()`, wall clock time

  `MPI_WTIME()`, for MPI codes

PDC Center for
High Performance Computing

B. Internal Timers

- Source code adapted to start, stop and save timers
- C calls:

  `gettimeofday()`, `time()` — time since January 1, 1970

  `clock()`, approximation of processor time

  `MPI_Wtime()`, for MPI codes

- Fortran calls:

  `system_clock()`, wall clock time

  `MPI_WTIME()`, for MPI codes

\+ A first easy to use and available method to measure time

PDC Center for
High Performance Computing

B. Internal Timers

- Source code adapted to start, stop and save timers
- C calls:

  `gettimeofday()`, `time()` — time since January 1, 1970

  `clock()`, approximation of processor time

  `MPI_Wtime()`, for MPI codes

- Fortran calls:

  `system_clock()`, wall clock time

  `MPI_WTIME()`, for MPI codes

\+ A first easy to use and available method to measure time

\- Affects the program execution time

PDC Center for
High Performance Computing

B. Internal Timers

- Source code adapted to start, stop and save timers
- C calls:

  `gettimeofday()`, `time()` — time since January 1, 1970

  `clock()`, approximation of processor time

  `MPI_Wtime()`, for MPI codes

PDC Center for
High Performance Computing

- Fortran calls:

  `system_clock()`, wall clock time

  `MPI_WTIME()`, for MPI codes

+ A first easy to use and available method to measure time
- Affects the program execution time
- Limited resolution (ms)

C. Performance Counters

- Hardware counters — registers counting events in the processor

PDC Center for
High Performance Computing

C. Performance Counters

- Hardware counters — registers counting events in the processor
- Registered on *every* CPU

PDC Center for
High Performance Computing

C. Performance Counters

- Hardware counters — registers counting events in the processor
- Registered on *every* CPU
- Cycles (perfect time resolution)

PDC Center for
High Performance Computing

C. Performance Counters

- Hardware counters — registers counting events in the processor
- Registered on *every* CPU
- Cycles (perfect time resolution)
- Instruction count (completed, floating point, integer, load/store)

PDC Center for
High Performance Computing

C. Performance Counters

- Hardware counters — registers counting events in the processor
- Registered on *every* CPU
- Cycles (perfect time resolution)
- Instruction count (completed, floating point, integer, load/store)
- Branches (Taken/not taken, etc)

PDC Center for
High Performance Computing

C. Performance Counters

- Hardware counters — registers counting events in the processor
- Registered on *every* CPU
- Cycles (perfect time resolution)
- Instruction count (completed, floating point, integer, load/store)
- Branches (Taken/not taken, etc)
- Cache (Cache level hits/misses)

PDC Center for
High Performance Computing

C. Performance Counters

- Hardware counters — registers counting events in the processor
- Registered on *every* CPU
- Cycles (perfect time resolution)
- Instruction count (completed, floating point, integer, load/store)
- Branches (Taken/not taken, etc)
- Cache (Cache level hits/misses)

\+ Measured event counts are exact

PDC Center for
High Performance Computing

C. Performance Counters

- Hardware counters — registers counting events in the processor
- Registered on *every* CPU
- Cycles (perfect time resolution)
- Instruction count (completed, floating point, integer, load/store)
- Branches (Taken/not taken, etc)
- Cache (Cache level hits/misses)

\+ Measured event counts are exact

\+ Usually doesn't affect performance too much

KTH VETENSKAP OCH KONST
PDC Center for High Performance Computing

C. Performance Counters

- Hardware counters — registers counting events in the processor
- Registered on *every* CPU
- Cycles (perfect time resolution)
- Instruction count (completed, floating point, integer, load/store)
- Branches (Taken/not taken, etc)
- Cache (Cache level hits/misses)

+ Measured event counts are exact
+ Usually doesn't affect performance too much

! Amount of data possible to store limited by registers

PDC Center for
High Performance Computing

C. Performance Counters

- Hardware counters — registers counting events in the processor
- Registered on *every* CPU
- Cycles (perfect time resolution)
- Instruction count (completed, floating point, integer, load/store)
- Branches (Taken/not taken, etc)
- Cache (Cache level hits/misses)

\+ Measured event counts are exact

\+ Usually doesn't affect performance too much

! Amount of data possible to store limited by registers

\- Requires CPU and OS support

PDC Center for High Performance Computing

C. Performance Counters

- Hardware counters — registers counting events in the processor
- Registered on *every* CPU
- Cycles (perfect time resolution)
- Instruction count (completed, floating point, integer, load/store)
- Branches (Taken/not taken, etc)
- Cache (Cache level hits/misses)

\+ Measured event counts are exact

\+ Usually doesn't affect performance too much

! Amount of data possible to store limited by registers

\- Requires CPU and OS support

\- Usually doesn't say where the problem is

KTH VETENSKAP OCH KONST

PDC Center for High Performance Computing

What do we want to know?

- **Where** does the code spend its time?

PDC Center for
High Performance Computing

What do we want to know?

- **Where** does the code spend its time?
- Want to know what the program actually does when run with a particular input data

PDC Center for
High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i − −
done()
```

PDC Center for
High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:  call init()
m2:  while i > 0
m3:  call calc()
m4:  i - -
m5:  call done()
```

PDC Center for
High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:  call init()
m2:  while i > 0
m3:  call calc()
m4:  i - -
m5:  call done()
```

```
i1:  a=0
i2:  b=10
i3:  i=4
```

PDC Center for High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:  call init()
m2:  while i > 0
m3:  call calc()
m4:  i - -
m5:  call done()
```

```
i1:  a=0
i2:  b=10
i3:  i=4
```

```
c1:  a=a+b
```

PDC Center for High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:   call init()
m2:   while i > 0
m3:   call calc()
m4:   i - -
m5:   call done()
```

```
i1:   a=0
i2:   b=10
i3:   i=4
```

```
c1:   a=a+b
```

```
d1:   print a
```

PDC Center for
High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:  call init()
m2:  while i > 0
m3:  call calc()
m4:  i - -
m5:  call done()
```

```
i1:  a=0
i2:  b=10
i3:  i=4
```

```
c1:  a=a+b
```

```
d1:  print a
```

```
m1
```

PDC Center for
High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:  call init()
m2:  while i > 0
m3:  call calc()
m4:  i - -
m5:  call done()
```

```
i1:  a=0
i2:  b=10
i3:  i=4
```

```
c1:  a=a+b
```

```
d1:  print a
```

```
m1
i1
```

PDC Center for High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:   call init()
m2:   while i > 0
m3:   call calc()
m4:   i - -
m5:   call done()
```

```
i1:   a=0
i2:   b=10
i3:   i=4
```

```
c1:   a=a+b
```

```
d1:   print a
```

```
m1
i1
i2
```

KTH VETENSKAP OCH KONST
PDC Center for High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:  call init()
m2:  while i > 0
m3:  call calc()
m4:  i - -
m5:  call done()
```

```
i1:  a=0
i2:  b=10
i3:  i=4
```

```
c1:  a=a+b
```

```
d1:  print a
```

```
m1
i1
i2
i3
```

PDC Center for High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:   call init()
m2:   while i > 0
m3:   call calc()
m4:   i - -
m5:   call done()
```

```
i1:   a=0
i2:   b=10
i3:   i=4
```

```
c1:   a=a+b
```

```
d1:   print a
```

```
m1
i1
i2
i3
m2
```

PDC Center for
High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:  call init()
m2:  while i > 0
m3:  call calc()
m4:  i - -
m5:  call done()
```

```
i1:  a=0
i2:  b=10
i3:  i=4
```

```
c1:  a=a+b
```

```
d1:  print a
```

```
m1
i1
i2
i3
m2
m3
```

PDC Center for High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:   call init()
m2:   while i > 0
m3:   call calc()
m4:   i - -
m5:   call done()
```

```
i1:   a=0
i2:   b=10
i3:   i=4
```

```
c1:   a=a+b
```

```
d1:   print a
```

```
m1
i1
i2
i3
m2
m3
c1
```

PDC Center for
High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:  call init()
m2:  while i > 0
m3:  call calc()
m4:  i - -
m5:  call done()
```

```
i1:  a=0
i2:  b=10
i3:  i=4
```

```
c1:  a=a+b
```

```
d1:  print a
```

```
m1
i1
i2
i3
m2
m3
c1
m4
```

PDC Center for
High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:  call init()
m2:  while i > 0
m3:  call calc()
m4:  i - -
m5:  call done()
```

```
i1:  a=0
i2:  b=10
i3:  i=4
```

```
c1:  a=a+b
```

```
d1:  print a
```

```
m1
i1
i2
i3
m2
m3
c1
m4
m2
```

Execution example

```
init()
while i>0
 calc()
 i --
done()
```

```
m1:  call init()
m2:  while i > 0
m3:  call calc()
m4:  i --
m5:  call done()
```

```
i1:  a=0
i2:  b=10
i3:  i=4
```

```
c1:  a=a+b
```

```
d1:  print a
```

```
m1
i1
i2
i3
m2
m3
c1
m4
m2
m3
```

PDC Center for High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:  call init()
m2:  while i > 0
m3:  call calc()
m4:  i - -
m5:  call done()
```

```
i1:  a=0
i2:  b=10
i3:  i=4
```

```
c1:  a=a+b
```

```
d1:  print a
```

```
m1
i1
i2
i3
m2
m3
c1
m4
m2
m3
c1
```

PDC Center for High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:   call init()
m2:   while i > 0
m3:   call calc()
m4:   i - -
m5:   call done()
```

```
i1:   a=0
i2:   b=10
i3:   i=4
```

```
c1:   a=a+b
```

```
d1:   print a
```

```
m1
i1
i2
i3
m2
m3
c1
m4
m2
m3
c1
m4
```

PDC Center for High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:  call init()
m2:  while i > 0
m3:  call calc()
m4:  i - -
m5:  call done()
```

```
i1:  a=0
i2:  b=10
i3:  i=4
```

```
c1:  a=a+b
```

```
d1:  print a
```

```
m1    m2
i1
i2
i3
m2
m3
c1
m4
m2
m3
c1
m4
```

Execution example

PDC Center for High Performance Computing

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:  call init()
m2:  while i > 0
m3:  call calc()
m4:  i - -
m5:  call done()
```

```
i1:  a=0
i2:  b=10
i3:  i=4
```

```
c1:  a=a+b
```

```
d1:  print a
```

```
m1
i1
i2
i3
m2
m3
c1
m4
m2
m3
c1
m4
m2
m3
```

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:   call init()
m2:   while i > 0
m3:   call calc()
m4:   i - -
m5:   call done()
```

```
i1:   a=0
i2:   b=10
i3:   i=4
```

```
c1:   a=a+b
```

```
d1:   print a
```

```
m1    m2
i1    m3
i2    c1
i3
m2
m3
c1
m4
m2
m3
c1
m4
```

PDC Center for High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:  call init()
m2:  while i > 0
m3:  call calc()
m4:  i - -
m5:  call done()
```

```
i1:  a=0
i2:  b=10
i3:  i=4
```

```
c1:  a=a+b
```

```
d1:  print a
```

```
m1   m2
i1   m3
i2   c1
i3   m4
m2
m3
c1
m4
m2
m3
c1
m4
```

PDC Center for High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:  call init()
m2:  while i > 0
m3:  call calc()
m4:  i - -
m5:  call done()
```

```
i1:  a=0
i2:  b=10
i3:  i=4
```

```
c1:  a=a+b
```

```
d1:  print a
```

m1	m2
i1	m3
i2	c1
i3	m4
m2	m2
m3	
c1	
m4	
m2	
m3	
c1	
m4	

PDC Center for High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:  call init()
m2:  while i > 0
m3:  call calc()
m4:  i - -
m5:  call done()
```

```
i1:  a=0
i2:  b=10
i3:  i=4
```

```
c1:  a=a+b
```

```
d1:  print a
```

```
m1   m2
i1   m3
i2   c1
i3   m4
m2   m2
m3   m3
c1
m4
m2
m3
c1
m4
```

PDC Center for
High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:   call init()
m2:   while i > 0
m3:   call calc()
m4:   i - -
m5:   call done()
```

```
i1:   a=0
i2:   b=10
i3:   i=4
```

```
c1:   a=a+b
```

```
d1:   print a
```

```
m1   m2
i1   m3
i2   c1
i3   m4
m2   m2
m3   m3
c1   c1
m4
m2
m3
c1
m4
```

PDC Center for High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:   call init()
m2:   while i > 0
m3:   call calc()
m4:   i - -
m5:   call done()
```

```
i1:   a=0
i2:   b=10
i3:   i=4
```

```
c1:   a=a+b
```

```
d1:   print a
```

m1	m2
i1	m3
i2	c1
i3	m4
m2	m2
m3	m3
c1	c1
m4	m4
m2	
m3	
c1	
m4	

PDC Center for High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:  call init()
m2:  while i > 0
m3:  call calc()
m4:  i - -
m5:  call done()
```

```
i1:  a=0
i2:  b=10
i3:  i=4
```

```
c1:  a=a+b
```

```
d1:  print a
```

m1	m2
i1	m3
i2	c1
i3	m4
m2	m2
m3	m3
c1	c1
m4	m4
m2	m2
m3	
c1	
m4	

PDC Center for
High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:  call init()
m2:  while i > 0
m3:  call calc()
m4:  i - -
m5:  call done()
```

```
i1:  a=0
i2:  b=10
i3:  i=4
```

```
c1:  a=a+b
```

```
d1:  print a
```

```
m1   m2
i1   m3
i2   c1
i3   m4
m2   m2
m3   m3
c1   c1
m4   m4
m2   m2
m3   m5
c1
m4
```

PDC Center for
High Performance Computing

Execution example

```
init()
while i>0
 calc()
 i - -
done()
```

```
m1:  call init()
m2:  while i > 0
m3:  call calc()
m4:  i - -
m5:  call done()
```

```
i1:  a=0
i2:  b=10
i3:  i=4
```

```
c1:  a=a+b
```

```
d1:  print a
```

```
m1   m2
i1   m3
i2   c1
i3   m4
m2   m2
m3   m3
c1   c1
m4   m4
m2   m2
m3   m5
c1   d1
m4
```

Too much data

- A $2\,\mathrm{GHz}$ processor running for $10\,\mathrm{s}$

PDC Center for
High Performance Computing

Too much data

- A $2\,\mathrm{GHz}$ processor running for $10\,\mathrm{s}$
- One instruction typically $32\,\mathrm{bit}$, i.e. $4\,\mathrm{byte}$.

PDC Center for
High Performance Computing

Too much data

- A $2\,\mathrm{GHz}$ processor running for $10\,\mathrm{s}$
- One instruction typically $32\,\mathrm{bit}$, i.e. $4\,\mathrm{byte}$.
- Total data generated — $2 * 10 * 4 = 80\,\mathrm{GB}$!

PDC Center for
High Performance Computing

Too much data

- A $2\,\mathrm{GHz}$ processor running for $10\,\mathrm{s}$
- One instruction typically $32\,\mathrm{bit}$, i.e. $4\,\mathrm{byte}$.
- Total data generated — $2 * 10 * 4 = 80\,\mathrm{GB}$!
- The data input will affect how your program runs

PDC Center for
High Performance Computing

Too much data

- A $2\,\mathrm{GHz}$ processor running for $10\,\mathrm{s}$
- One instruction typically $32\,\mathrm{bit}$, i.e. $4\,\mathrm{byte}$.
- Total data generated — $2 * 10 * 4 = 80\,\mathrm{GB}$!
- The data input will affect how your program runs
- Code length will always be $\ll$ number of instructions executed!

PDC Center for
High Performance Computing

Too much data

- A $2\,\mathrm{GHz}$ processor running for $10\,\mathrm{s}$
- One instruction typically $32\,\mathrm{bit}$, i.e. $4\,\mathrm{byte}$.
- Total data generated — $2 * 10 * 4 = 80\,\mathrm{GB}$!
- The data input will affect how your program runs
- Code length will always be $\ll$ number of instructions executed!
- Need a way of reducing data and still get the information!

PDC Center for
High Performance Computing

D. Profilers

- Two types: **Statistical and Event based profilers**

PDC Center for
High Performance Computing

D. Profilers

- Two types: **Statistical and Event based profilers**
- Statistical Profiling:

PDC Center for
High Performance Computing

D. Profilers

- Two types: **Statistical and Event based profilers**
- Statistical Profiling:

  Interrupts at **random intervals** and records which program instruction the CPU is executing.

PDC Center for
High Performance Computing

D. Profilers

- Two types: **Statistical and Event based profilers**
- Statistical Profiling:

  Interrupts at **random intervals** and records which program instruction the CPU is executing.
- Event based Profiling:

PDC Center for
High Performance Computing

D. Profilers

- Two types: **Statistical and Event based profilers**
- Statistical Profiling:

  Interrupts at **random intervals** and records which program instruction the CPU is executing.
- Event based Profiling:

  Interrupts triggered by **hardware counter events** are recorded.

PDC Center for
High Performance Computing

D. Profilers

- Two types: **Statistical and Event based profilers**
- Statistical Profiling:

  Interrupts at **random intervals** and records which program instruction the CPU is executing.
- Event based Profiling:

  Interrupts triggered by **hardware counter events** are recorded.

\- Measuring profiles affects performance

PDC Center for
High Performance Computing

D. Profilers

- Two types: **Statistical and Event based profilers**
- Statistical Profiling:

  Interrupts at **random intervals** and records which program instruction the CPU is executing.
- Event based Profiling:

  Interrupts triggered by **hardware counter events** are recorded.

- Measuring profiles affects performance
- Still a lot of data saved

PDC Center for
High Performance Computing

E. Call Tracing

- Call Tracing — Library specific profiling (for example MPI)

PDC Center for
High Performance Computing

E. Call Tracing

- Call Tracing — Library specific profiling (for example MPI)
- Wrappers for library specific function calls (for example `MPI_SEND`)

PDC Center for
High Performance Computing

E. Call Tracing

- Call Tracing — Library specific profiling (for example MPI)
- Wrappers for library specific function calls (for example `MPI_SEND`)
- Records **when** a function was called and **with what parameters**

PDC Center for
High Performance Computing

E. Call Tracing

- Call Tracing — Library specific profiling (for example MPI)
- Wrappers for library specific function calls (for example `MPI_SEND`)
- Records **when** a function was called and **with what parameters**
- Get the whole picture — post processing

PDC Center for
High Performance Computing

E. Call Tracing

- Call Tracing — Library specific profiling (for example MPI)
- Wrappers for library specific function calls (for example `MPI_SEND`)
- Records **when** a function was called and **with what parameters**
- Get the whole picture — post processing

\+ Gives you library specific information — which nodes exchanged messages, what was the message size. . .

PDC Center for
High Performance Computing

E. Call Tracing

- Call Tracing — Library specific profiling (for example MPI)
- Wrappers for library specific function calls (for example `MPI_SEND`)
- Records **when** a function was called and **with what parameters**
- Get the whole picture — post processing

\+ Gives you library specific information — which nodes exchanged messages, what was the message size. . .

\- Affects performance (depending on how often library calls are made)

PDC Center for High Performance Computing

Performance Tools on Ferlin

Name:	Type:	License:
gprof	statistical profiler	free
papiex	performance counter	free/licensed
mpip	MPI profiling	free
paraver	MPI tracing and profiling	free
tau	MPI tracing and profiling	free
jumpshot	visualization of MPI traces	free

PDC Center for
High Performance Computing

Performance Tools on Ferlin

Name:	Type:	License:
gprof	statistical profiler	free
papiex	performance counter	free/licensed
mpip	MPI profiling	free
paraver	MPI tracing and profiling	free
tau	MPI tracing and profiling	free
jumpshot	visualization of MPI traces	free

PDC Center for
High Performance Computing

What is Good Parallel Performance?

- Single CPU performance is high.

PDC Center for
High Performance Computing

What is Good Parallel Performance?

- Single CPU performance is high.
- The code is scalable out to more than a few nodes.

PDC Center for
High Performance Computing

What is Good Parallel Performance?

- Single CPU performance is high.
- The code is scalable out to more than a few nodes.
- The network is not the bottleneck.

PDC Center for
High Performance Computing

What is Good Parallel Performance?

- Single CPU performance is high.
- The code is scalable out to more than a few nodes.
- The network is not the bottleneck.
- Data sent around is at a minimum

PDC Center for
High Performance Computing

What is Good Parallel Performance?

- Single CPU performance is high.
- The code is scalable out to more than a few nodes.
- The network is not the bottleneck.
- Data sent around is at a minimum
- Use Performance Tools to get there!

PDC Center for
High Performance Computing