Resurrecting GSLIB by code optimization and multi-core programming

Oscar Peredo[1] & Julián M. Ortiz[1,2]

[1] Advanced Lab for Geostatistical Supercomputing, Advanced Mining Technology Center, University of Chile

[2] Department of Mining Engineering, University of Chile

16th Annual Conference of the International Association for Mathematical Geosciences,

17-20 October, 2014 - New Delhi, India

Outline

Outline

Motivation

- GSLIB: open-source software package (*applications* + *utilities*) used in the geostatistical community for more than 30 years.

Motivation

- GSLIB: open-source software package (*applications* + *utilities*) used in the geostatistical community for more than 30 years.

- Significant efforts have been developed to accelerate/enhance the scope of the original package (SGEMS, WinGSLIB), but according to the authors' knowledge, few efforts have been reported in order to accelerate the GSLIB package by itself.

Motivation

- GSLIB: open-source software package (*applications* + *utilities*) used in the geostatistical community for more than 30 years.

- Significant efforts have been developed to accelerate/enhance the scope of the original package (SGEMS, WinGSLIB), but according to the authors' knowledge, few efforts have been reported in order to accelerate the GSLIB package by itself.

- How can we accelerate an application *by itself*?

Optimizing its code and using multiple threads (cores) of execution

What is *optimize code*?

- *Code optimization*: set of techniques and code modifications (machine-independent and machine-dependent) that can help us to accelerate an application using a general-purpose CPU.

What is *optimize code*?

- *Code optimization*: set of techniques and code modifications (machine-independent and machine-dependent) that can help us to accelerate an application using a general-purpose CPU.

- Examples:

What is *optimize code*?

- *Code optimization*: set of techniques and code modifications (machine-independent and machine-dependent) that can help us to accelerate an application using a general-purpose CPU.

- Examples:
  - *Memoization*

What is *optimize code*?

- *Code optimization*: set of techniques and code modifications (machine-independent and machine-dependent) that can help us to accelerate an application using a general-purpose CPU.

- Examples:
  - *Memoization*
  - Inlining

What is *optimize code*?

- *Code optimization*: set of techniques and code modifications (machine-independent and machine-dependent) that can help us to accelerate an application using a general-purpose CPU.

- Examples:
  - *Memoization*
  - Inlining
  - Reduction of miss-predicted braches

What is *optimize code*?

- *Code optimization*: set of techniques and code modifications (machine-independent and machine-dependent) that can help us to accelerate an application using a general-purpose CPU.

- Examples:
  - *Memoization*
  - Inlining
  - Reduction of miss-predicted braches
  - Loop unrolling

What is *optimize code*?

- *Code optimization*: set of techniques and code modifications (machine-independent and machine-dependent) that can help us to accelerate an application using a general-purpose CPU.

- Examples:
  - *Memoization*
  - Inlining
  - Reduction of miss-predicted braches
  - Loop unrolling
  - Data locality and memory bandwidth

What is *optimize code*?

- *Code optimization*: set of techniques and code modifications (machine-independent and machine-dependent) that can help us to accelerate an application using a general-purpose CPU.

- Examples:
  - *Memoization*
  - Inlining
  - Reduction of miss-predicted braches
  - Loop unrolling
  - Data locality and memory bandwidth
  - SIMD programming

What is *optimize code*?

- *Code optimization*: set of techniques and code modifications (machine-independent and machine-dependent) that can help us to accelerate an application using a general-purpose CPU.

- Examples:
  - *Memoization*
  - Inlining
  - Reduction of miss-predicted braches
  - Loop unrolling
  - Data locality and memory bandwidth
  - SIMD programming
  - ...

What is *optimize code*?

- *Code optimization*: set of techniques and code modifications (machine-independent and machine-dependent) that can help us to accelerate an application using a general-purpose CPU.

- Examples:
  - *Memoization*
  - Inlining
  - Reduction of miss-predicted braches
  - Loop unrolling
  - Data locality and memory bandwidth
  - SIMD programming
  - ...

What is *optimize code*?

- *Code optimization*: set of techniques and code modifications (machine-independent and machine-dependent) that can help us to accelerate an application using a general-purpose CPU.

- Examples:
  - *Memoization*
  - Inlining
  - Reduction of miss-predicted braches
  - Loop unrolling
  - Data locality and memory bandwidth
  - SIMD programming
  - ...

$$\text{GSLIB} + \text{Code optimization}$$

$$=$$

faster GSLIB!

What is *multi-core programming*?

- *Multi-core programming*: set of techniques and programming models that can help us to accelerate an application using multiple general-purpose CPUs, typically sharing the same memory address space (intra-node communication).

What is *multi-core programming*?

- *Multi-core programming*: set of techniques and programming models that can help us to accelerate an application using multiple general-purpose CPUs, typically sharing the same memory address space (intra-node communication).

- Examples:

What is *multi-core programming*?

- *Multi-core programming*: set of techniques and programming models that can help us to accelerate an application using multiple general-purpose CPUs, typically sharing the same memory address space (intra-node communication).

- Examples:
  - OpenMP

What is *multi-core programming*?

- *Multi-core programming*: set of techniques and programming models that can help us to accelerate an application using multiple general-purpose CPUs, typically sharing the same memory address space (intra-node communication).

- Examples:
  - OpenMP
  - POSIX threads

What is *multi-core programming*?

- *Multi-core programming*: set of techniques and programming models that can help us to accelerate an application using multiple general-purpose CPUs, typically sharing the same memory address space (intra-node communication).

- Examples:
  - OpenMP
  - POSIX threads
  - MPI (intra-node)

What is *multi-core programming*?

- *Multi-core programming*: set of techniques and programming models that can help us to accelerate an application using multiple general-purpose CPUs, typically sharing the same memory address space (intra-node communication).

- Examples:
  - OpenMP
  - POSIX threads
  - MPI (intra-node)
  - Intel's TBB

What is *multi-core programming*?

- *Multi-core programming*: set of techniques and programming models that can help us to accelerate an application using multiple general-purpose CPUs, typically sharing the same memory address space (intra-node communication).

- Examples:
  - OpenMP
  - POSIX threads
  - MPI (intra-node)
  - Intel's TBB
  - Erlang (intra-node)

What is *multi-core programming*?

- *Multi-core programming*: set of techniques and programming models that can help us to accelerate an application using multiple general-purpose CPUs, typically sharing the same memory address space (intra-node communication).

- Examples:
  - OpenMP
  - POSIX threads
  - MPI (intra-node)
  - Intel's TBB
  - Erlang (intra-node)
  - ...

What is *multi-core programming*?

- *Multi-core programming*: set of techniques and programming models that can help us to accelerate an application using multiple general-purpose CPUs, typically sharing the same memory address space (intra-node communication).

- Examples:
  - OpenMP
  - POSIX threads
  - MPI (intra-node)
  - Intel's TBB
  - Erlang (intra-node)
  - ...

What is *multi-core programming*?

- *Multi-core programming*: set of techniques and programming models that can help us to accelerate an application using multiple general-purpose CPUs, typically sharing the same memory address space (intra-node communication).

- Examples:
  - OpenMP
  - POSIX threads
  - MPI (intra-node)
  - Intel's TBB
  - Erlang (intra-node)
  - ...

$$\text{GSLIB} + \text{Code optimization} + \text{Multi-core programming}$$

$$=$$

even faster GSLIB!!!

Objective

Develop a new open-source version of GSLIB, that uses optimized code and multiple threads (cores) of execution.

Outline

Methodology

1. Pick an application and re-design it for acceleration.

Methodology

1. Pick an application and re-design it for acceleration.
2. Create a run-time profile, identifying bottle-necks and optimize those bottle-necks.

Methodology

1. Pick an application and re-design it for acceleration.
2. Create a run-time profile, identifying bottle-necks and optimize those bottle-necks.
3. Adapt the application for multi-thread execution.

Methodology

1. Pick an application and re-design it for acceleration.
2. Create a run-time profile, identifying bottle-necks and optimize those bottle-necks.
3. Adapt the application for multi-thread execution.

Methodology

1. Pick an application and re-design it for acceleration.
2. Create a run-time profile, identifying bottle-necks and optimize those bottle-necks.
3. Adapt the application for multi-thread execution.

Let's see these steps using `gamv` as example...

Outline

Step 1: Re-design

- Inline important subroutines (`readparm` and `gamv` in this case):

```
subrountine foo(a)
   integer a
   a=a+2
end subroutine foo

program main
   integer a,b
   a=1
   call foo(a)
   b=a
end program main
```

```
csubrountine foo(a)
c   integer a
c   a=a+2
cend subroutine foo

program main
   integer a,b
   a=1
c   call foo(a)
   b=a+2
end program main
```

Step 1: Re-design

- Inline important subroutines (`readparm` and `gamv` in this case):

```
subrountine foo(a)
   integer a
   a=a+2
end subroutine foo

program main
   integer a,b
   a=1
   call foo(a)
   b=a
end program main
```

```
csubrountine foo(a)
c   integer a
c   a=a+2
cend subroutine foo

program main
   integer a,b
   a=1
c   call foo(a)
   b=a+2
end program main
```

- Move global variables from `module geostat` and common blocks (`include file.inc`) to main program.

Step 1: Re-design

- Inline important subroutines (`readparm` and `gamv` in this case):

```
subrountine foo(a)
   integer a
   a=a+2
end subroutine foo

program main
   integer a,b
   a=1
   call foo(a)
   b=a
end program main
```

```
csubrountine foo(a)
c   integer a
c   a=a+2
cend subroutine foo

program main
   integer a,b
   a=1
c   call foo(a)
   b=a+2
end program main
```

- Move global variables from `module geostat` and common blocks (`include file.inc`) to main program.
- Use `implicit none` in all non-inlined subroutines.

Step 1: Re-design

- Inline important subroutines (`readparm` and `gamv` in this case):

```
subrountine foo(a)
   integer a
   a=a+2
end subroutine foo

program main
   integer a,b
   a=1
   call foo(a)
   b=a
end program main
```

```
csubrountine foo(a)
c   integer a
c   a=a+2
cend subroutine foo

program main
   integer a,b
   a=1
c   call foo(a)
   b=a+2
end program main
```

- Move global variables from `module geostat` and common blocks (`include file.inc`) to main program.
- Use `implicit none` in all non-inlined subroutines.
- Avoid side-effects in each subroutine, passing all necessary data as explicit subroutine parameters.

Step 1: Re-design

- Inline important subroutines (`readparm` and `gamv` in this case):

```
subrountine foo(a)
   integer a
   a=a+2
end subroutine foo

program main
   integer a,b
   a=1
   call foo(a)
   b=a
end program main
```

```
csubrountine foo(a)
c   integer a
c   a=a+2
cend subroutine foo

program main
   integer a,b
   a=1
c   call foo(a)
   b=a+2
end program main
```

- Move global variables from `module geostat` and common blocks (`include file.inc`) to main program.
- Use `implicit none` in all non-inlined subroutines.
- Avoid side-effects in each subroutine, passing all necessary data as explicit subroutine parameters.

Step 1: Re-design

- Inline important subroutines (`readparm` and `gamv` in this case):

```
subrountine foo(a)
   integer a
   a=a+2
end subroutine foo

program main
   integer a,b
   a=1
   call foo(a)
   b=a
end program main
```

```
csubrountine foo(a)
c   integer a
c   a=a+2
cend subroutine foo

program main
   integer a,b
   a=1
c   call foo(a)
   b=a+2
end program main
```

- Move global variables from `module geostat` and common blocks (`include file.inc`) to main program.
- Use `implicit none` in all non-inlined subroutines.
- Avoid side-effects in each subroutine, passing all necessary data as explicit subroutine parameters.

Maybe the re-designed code doesn't look pretty, but the ***data-flow path* of each variable can be easily identified (important for multi-core programming).**

Example: gamv

Original code:	Re-designed code:

module geostat
  real, allocatable::
  x(:),y(:),z(:)
  ...
end module geostat
program main
  use geostat
  call readparm
  call gamv
  call writeout
  stop
  end
end program main
	program main
	c move module vars into main
	  real, allocatable::
	  x(:),y(:),z(:)
	  ...
	c inlined code of readparm
	  ...
	c inlined code of gamv
	  ...
	c writeout is not inlined
	  call writeout
	  stop
	  end
	end program main

Outline

Step 2: Profiling and code optimization

- Compile the code using the *debug* flag (`-g`).

Step 2: Profiling and code optimization

- Compile the code using the *debug* flag (`-g`).
- Select a Fortran profiler tool (well-known tools for Linux: `gprof` and `oprofile`).

Step 2: Profiling and code optimization

- Compile the code using the *debug* flag (`-g`).
- Select a Fortran profiler tool (well-known tools for Linux: `gprof` and `oprofile`).
- Identify bottle-necks looking at the performance profiles obtained (sampling in hardware events).

Step 2: Profiling and code optimization

- Compile the code using the *debug* flag (`-g`).
- Select a Fortran profiler tool (well-known tools for Linux: `gprof` and `oprofile`).
- Identify bottle-necks looking at the performance profiles obtained (sampling in hardware events).
- Optimize the code removing or minimizing those bottle-necks.

Example: `oprofile` output over `gamv`

Event=`BR_MISS_PRED_RETIRED`, sampling freq. 500 hw events.

```
 #events %total : line of code
-------------------------------------------------------------------------------
                :...
                :            if(hs.gt.dismxs) go to 4
  1663 14.7954 :            if(hs.lt.0.0) hs = 0.0
     1  0.0089 :            h    = sqrt(hs)
                :c
                :c Determine which lag this is and skip if outside the defined
                :c distance tolerance:
                :c
                :            if(h.le.EPSLON) then
                :                  lagbeg = 1
     1  0.0089 :                  lagend = 1
                :            else
                :                  lagbeg = -1
                :                  lagend = -1
  2510 22.3310 :                  do ilag=2,nlag+2
  3813 33.9235 :                        if(h.ge.(xlag*real(ilag-2)-xltol).and.
                :     +                    h.le.(xlag*real(ilag-2)+xltol)) then
   254  2.2598 :                           if(lagbeg.lt.0) lagbeg = ilag
     4  0.0356 :                           lagend = ilag
                :                     end if
                :                  end do
     1  0.0089 :                  if(lagend.lt.0) go to 4
                :            endif
                :...
```

Example: `oprofile` output over `gamv`

Event=`BR_MISS_PRED_RETIRED`, sampling freq. 500 hw events.

```
  #events %total : line of code
 ------------------------------------------------------------------------------
                 :...
                 :            if(hs.gt.dismxs) go to 4
    1663 14.7954 :            if(hs.lt.0.0) hs = 0.0
       1  0.0089 :            h    = sqrt(hs)
                 :c
                 :c Determine which lag this is and skip if outside the defined
                 :c distance tolerance:
                 :c
                 :            if(h.le.EPSLON) then
                 :                  lagbeg = 1
       1  0.0089 :                  lagend = 1
                 :            else
                 :                  lagbeg = -1
                 :                  lagend = -1
    2510 22.3310 :                  do ilag=2,nlag+2
    3813 33.9235 :                        if(h.ge.(xlag*real(ilag-2)-xltol).and.
                 :     +                     h.le.(xlag*real(ilag-2)+xltol)) then
     254  2.2598 :                           if(lagbeg.lt.0) lagbeg = ilag
       4  0.0356 :                           lagend = ilag
                 :                     end if
                 :                  end do
       1  0.0089 :                  if(lagend.lt.0) go to 4
                 :            endif
                 :...
```

Too many "`if`" instructions

Example: `oprofile` output over `gamv` optimized

Event=`BR_MISS_PRED_RETIRED`, sampling freq. 500 hw events.

```
 #events %total : line of code
-------------------------------------------------------------------------------
                :...
                :            if(hs.gt.dismxs) go to 4
  1452 17.8356 :            if(hs.lt.0.0) hs = 0.0
     2  0.0246 :            h    = sqrt(hs)
                :c
                :c Determine which lag this is and skip if outside the defined
                :c distance tolerance:
                :c
                :            if(h.le.EPSLON) then
                :                  lagbeg = 1
                :                  lagend = 1
                :            else
                :                  lagbeg = -1
                :                  lagend = -1
                :                  liminf=(ceiling((h-xltol)*xlaginv)+2)
   270  3.3165 :                  limsup=(floor((h+xltol)*xlaginv)+2)
  1022 12.5537 :                  do ilag=liminf,limsup
   700  8.5985 :                        if(lagbeg.lt.0) lagbeg = ilag
   350  4.2992 :                        lagend = ilag
                :
                :                  end do
   350  4.2992 :                  if(lagend.lt.0) go to 4
                :            endif
                :...
```

Example: `oprofile` output over `gamv` optimized

Event=`BR_MISS_PRED_RETIRED`, sampling freq. 500 hw events.

```
  #events %total : line of code
-------------------------------------------------------------------------------
                 :...
                 :            if(hs.gt.dismxs) go to 4
   1452 17.8356 :            if(hs.lt.0.0) hs = 0.0
      2  0.0246 :            h    = sqrt(hs)
                 :c
                 :c Determine which lag this is and skip if outside the defined
                 :c distance tolerance:
                 :c
                 :            if(h.le.EPSLON) then
                 :                  lagbeg = 1
                 :                  lagend = 1
                 :            else
                 :                  lagbeg = -1
                 :                  lagend = -1
                 :                  liminf=(ceiling((h-xltol)*xlaginv)+2)
    270  3.3165 :                  limsup=(floor((h+xltol)*xlaginv)+2)
   1022 12.5537 :                  do ilag=liminf,limsup
    700  8.5985 :                        if(lagbeg.lt.0) lagbeg = ilag
    350  4.2992 :                        lagend = ilag
                 :
                 :                  end do
    350  4.2992 :                  if(lagend.lt.0) go to 4
                 :            endif
                 :...
```

Acceleration 6%

Example: `oprofile` output over `gamv`

Event=`L1D_BLOCKS`, sampling freq. 100000 hw events.

```
 #events %total : line of code
-------------------------------------------------------------------------------
               :...
               :c MAIN LOOP OVER ALL PAIRS:
               :c
               :      irepo = max(1,min((nd/10),1000))
               :      do 3 i=1,nd
               :        if((int(i/irepo)*irepo).eq.i) write(*,103) i,nd
               :
               :
               :
  488  2.7767 :      do 4 j=i,nd
               :c
               :c Definition of the lag corresponding to the current pair:
               :c
 1500  8.5349 :            dx  = x(j) - x(i)
  901  5.1266 :            dy  = y(j) - y(i)
 1666  9.4794 :            dz  = z(j) - z(i)
   88  0.5007 :            dxs = dx*dx
  811  4.6145 :            dys = dy*dy
 1964 11.1750 :            dzs = dz*dz
 5610 31.9203 :            hs  = dxs + dys + dzs
 3837 21.8321 :            if(hs.gt.dismxs) go to 4
    8  0.0455 :            if(hs.lt.0.0) hs = 0.0
    2  0.0114 :            h   = sqrt(hs)
               :...
```

Example: `oprofile` output over `gamv`

Event=`L1D_BLOCKS`, sampling freq. 100000 hw events.

```
 #events %total : line of code
-------------------------------------------------------------------------------
                :...
                :c MAIN LOOP OVER ALL PAIRS:
                :c
                :      irepo = max(1,min((nd/10),1000))
                :      do 3 i=1,nd
                :        if((int(i/irepo)*irepo).eq.i) write(*,103) i,nd
                :
                :
                :
   488  2.7767 :      do 4 j=i,nd
                :c
                :c Definition of the lag corresponding to the current pair:
                :c
  1500  8.5349 :            dx  = x(j) - x(i)
   901  5.1266 :            dy  = y(j) - y(i)
  1666  9.4794 :            dz  = z(j) - z(i)
    88  0.5007 :            dxs = dx*dx
   811  4.6145 :            dys = dy*dy
  1964 11.1750 :            dzs = dz*dz
  5610 31.9203 :            hs  = dxs + dys + dzs
  3837 21.8321 :            if(hs.gt.dismxs) go to 4
     8  0.0455 :            if(hs.lt.0.0) hs = 0.0
     2  0.0114 :            h   = sqrt(hs)
                :...
```

Too many memory accesses

Example: `oprofile` output over `gamv` optimized

Event=`L1D_BLOCKS`, sampling freq. 100000 hw events.

```
 #events %total : line of code
-------------------------------------------------------------------------------
               :...
               :c MAIN LOOP OVER ALL PAIRS:
               :c
               :      irepo = max(1,min((nd/10),1000))
               :      do 3 i=1,nd
               :        if((int(i/irepo)*irepo).eq.i) write(*,103) i,nd
               :            xi=x(i)
               :            yi=y(i)
               :            zi=z(i)
   291  1.3619 :      do 4 j=i,nd
               :c
               :c Definition of the lag corresponding to the current pair:
               :c
   802  3.7535 :            dx  = x(j) - xi
  1040  4.8673 :            dy  = y(j) - yi
  1368  6.4024 :            dz  = z(j) - zi
   785  3.6739 :            dxs = dx*dx
   595  2.7847 :            dys = dy*dy
   148  0.6927 :            dzs = dz*dz
  5881 27.5238 :            hs  = dxs + dys + dzs
  3574 16.7267 :            if(hs.gt.dismxs) go to 4
     6  0.0281 :            if(hs.lt.0.0) hs = 0.0
               :            h   = sqrt(hs)
               :...
```

Example: `oprofile` output over `gamv` optimized

Event=`L1D_BLOCKS`, sampling freq. 100000 hw events.

```
 #events %total : line of code
-------------------------------------------------------------------------------
                :...
                :c MAIN LOOP OVER ALL PAIRS:
                :c
                :      irepo = max(1,min((nd/10),1000))
                :      do 3 i=1,nd
                :        if((int(i/irepo)*irepo).eq.i) write(*,103) i,nd
                :            xi=x(i)
                :            yi=y(i)
                :            zi=z(i)
   291  1.3619 :      do 4 j=i,nd
                :c
                :c Definition of the lag corresponding to the current pair:
                :c
   802  3.7535 :            dx  = x(j) - xi
  1040  4.8673 :            dy  = y(j) - yi
  1368  6.4024 :            dz  = z(j) - zi
   785  3.6739 :            dxs = dx*dx
   595  2.7847 :            dys = dy*dy
   148  0.6927 :            dzs = dz*dz
  5881 27.5238 :            hs  = dxs + dys + dzs
  3574 16.7267 :            if(hs.gt.dismxs) go to 4
     6  0.0281 :            if(hs.lt.0.0) hs = 0.0
                :            h   = sqrt(hs)
                :...
```

Acceleration 22%

Outline

Step 3: Multi-core execution

- We choose the OpenMP programming model: minimal modifications in the original code and easy implementation.

Step 3: Multi-core execution

- We choose the OpenMP programming model: minimal modifications in the original code and easy implementation.
- Compile the code using the openmp flag (`-fopenmp` for GCC or `-openmp` for Intel compilers).

Step 3: Multi-core execution

- We choose the OpenMP programming model: minimal modifications in the original code and easy implementation.
- Compile the code using the openmp flag (`-fopenmp` for GCC or `-openmp` for Intel compilers).
- Identify the most time consuming parts of the code, looking the previous performance profiles obtained.

Step 3: Multi-core execution

- We choose the OpenMP programming model: minimal modifications in the original code and easy implementation.
- Compile the code using the openmp flag (`-fopenmp` for GCC or `-openmp` for Intel compilers).
- Identify the most time consuming parts of the code, looking the previous performance profiles obtained.
- Add OpenMP directives in order to parallelize those parts using multiple threads of execution (ideally, 1 thread must run in 1 CPU core).

Example: +100% faster `gamv`

```
...
c
c MAIN LOOP OVER ALL PAIRS:
c
      irepo = max(1,min((nd/10),1000))









      do 3 i=1,nd
        if((int(i/irepo)*irepo).eq.i) write(*,103) i,nd
            xi=x(i)
            yi=y(i)
            zi=z(i)
      do 4 j=i,nd
c
c Definition of the lag corresponding to the current pair:
c
            dx  = x(j) - xi
            dy  = y(j) - yi
            dz  = z(j) - zi
            dxs = dx*dx
            dys = dy*dy
            dzs = dz*dz
...


...
```

Example: +100% faster `gamv`

```
...
c
c MAIN LOOP OVER ALL PAIRS:
c
      irepo = max(1,min((nd/10),1000))




      do 3 i=1,nd
        if((int(i/irepo)*irepo).eq.i) write(*,103) i,nd
            xi=x(i)
            yi=y(i)
            zi=z(i)
      do 4 j=i,nd
c
c Definition of the lag corresponding to the current pair:
c
            dx  = x(j) - xi
            dy  = y(j) - yi
            dz  = z(j) - zi
            dxs = dx*dx
            dys = dy*dy
            dzs = dz*dz
...


...
```

$$\frac{nd*(nd+1)}{2}$$ pairs of points to process

Example: +100% faster `gamv` + OpenMP

```
...
c
c MAIN LOOP OVER ALL PAIRS:
c
      irepo = max(1,min((nd/10),1000))
c$omp parallel default(firstprivate) shared(x,y,z,reducedVariables)
#ifdef _OPENMP
      threadId = int(OMP_get_thread_num())+1
#endif
c$omp do schedule(runtime)
      do 3 i=1,nd
        if((int(i/irepo)*irepo).eq.i) write(*,103) i,nd
            xi=x(i)
            yi=y(i)
            zi=z(i)
      do 4 j=i,nd
c
c Definition of the lag corresponding to the current pair:
c
            dx  = x(j) - xi
            dy  = y(j) - yi
            dz  = z(j) - zi
            dxs = dx*dx
            dys = dy*dy
            dzs = dz*dz
...
c$ omp end do
c$ omp end parallel
...
```

Example: +100% faster `gamv` + OpenMP

```
...
c
c MAIN LOOP OVER ALL PAIRS:
c
      irepo = max(1,min((nd/10),1000))
c$omp parallel default(firstprivate) shared(x,y,z,reducedVariables)
#ifdef _OPENMP
      threadId = int(OMP_get_thread_num())+1
#endif
c$omp do schedule(runtime)
      do 3 i=1,nd
        if((int(i/irepo)*irepo).eq.i) write(*,103) i,nd
            xi=x(i)
            yi=y(i)
            zi=z(i)
      do 4 j=i,nd
c
c Definition of the lag corresponding to the current pair:
c
            dx  = x(j) - xi
            dy  = y(j) - yi
            dz  = z(j) - zi
            dxs = dx*dx
            dys = dy*dy
            dzs = dz*dz
...
c$ omp end do
c$ omp end parallel
...
```

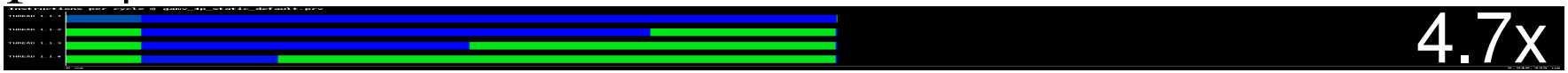
Outer-loop is divided among T threads using a load-balance policy defined at runtime

Example: +100% faster `gamv` + OpenMP

Runtime policy: "`static`" (default)
Each thread process `nd`/T contiguous iterations

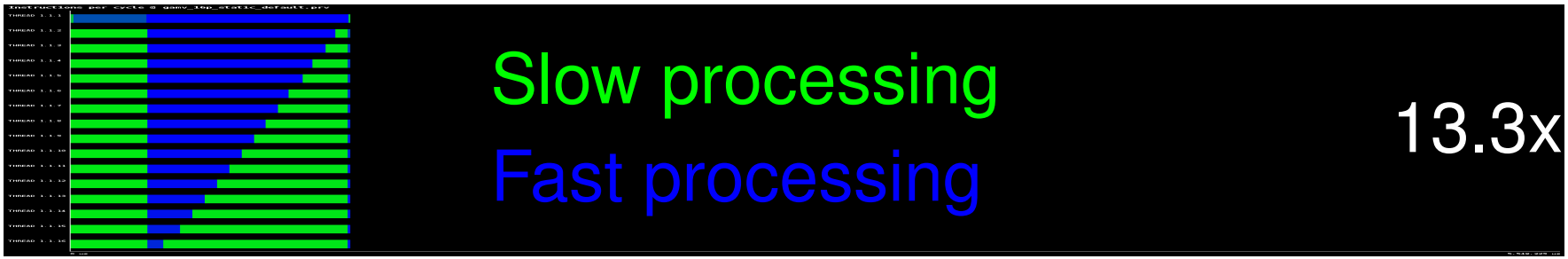
$T = 2$

$T = 4$



$T = 8$

$T = 16$



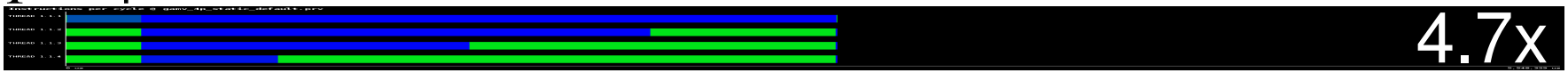
Can we run faster?

Example: +100% faster `gamv` + OpenMP

Runtime policy: "`static`" (default)
Each thread process `nd`/T contiguous iterations

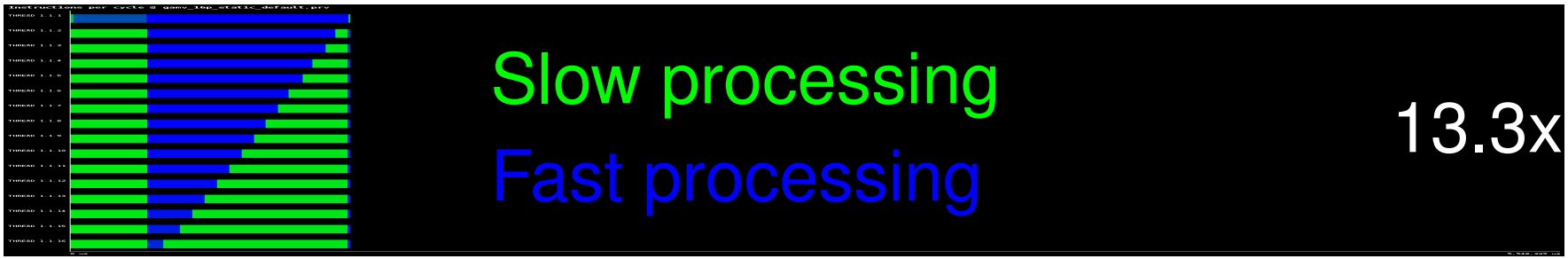
$T = 2$

$T = 4$



$T = 8$

$T = 16$



Can we run faster?

Example: +100% faster `gamv` + OpenMP

Runtime policy: "`static,16`"

Each thread process 16 contiguous iterations (interleaved)

$T = 2$

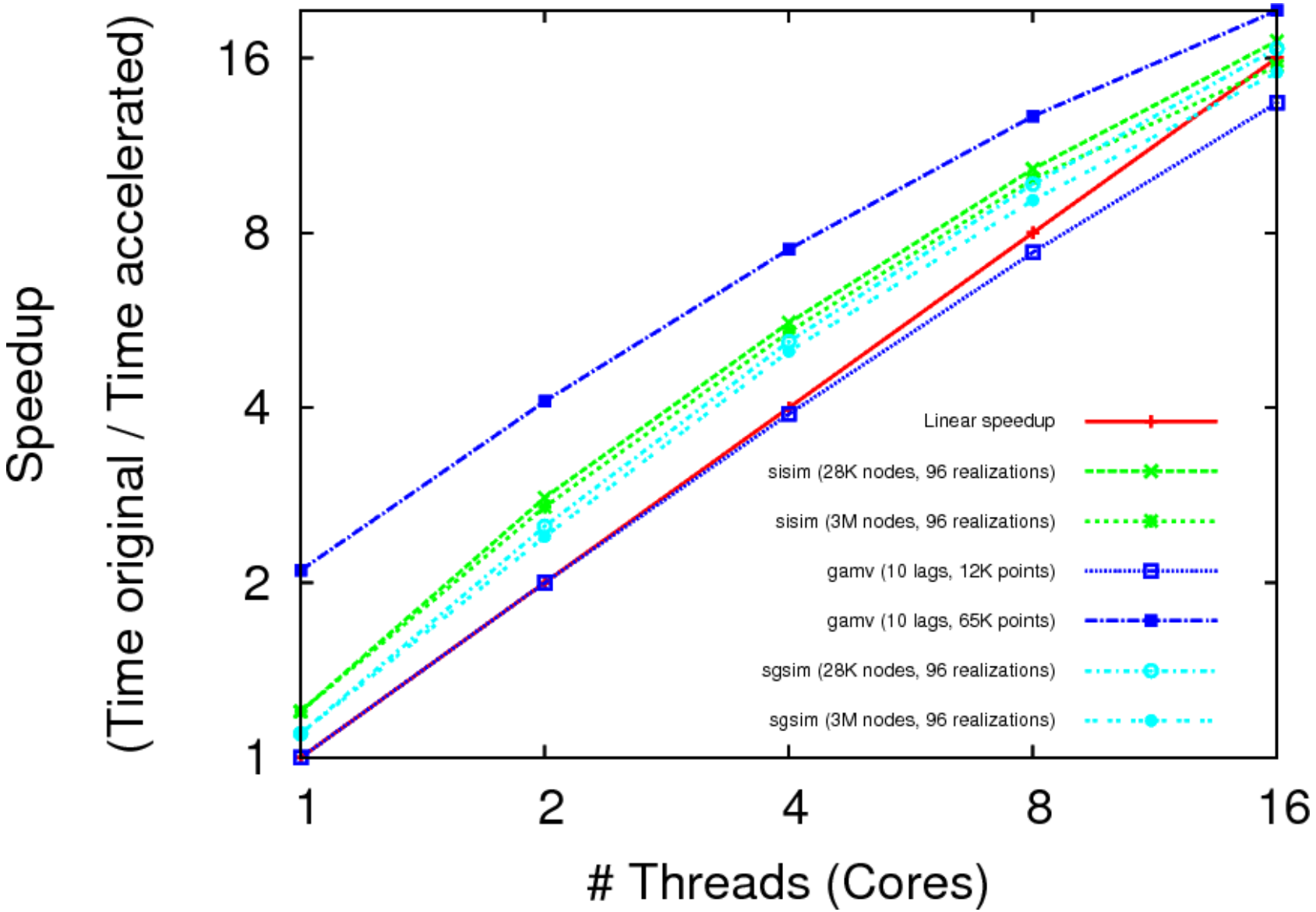
$T = 4$

$T = 8$

$T = 16$

Yes!

Other results: `gamv`, `sisim` and `sgsim`



Outline

Conclusions

- **The proposed methodology accelerates the runtime execution of GSLIB applications, without changing the classical usability (parameter files).**
- Many code optimization techniques can be applied to the target code, in a combined way.
- Different speedup results can be obtained using different load-balancing policies, so we must choose the best for our problem.
- Current developments: `gamv`, `sisim` and `sgsim`
  - Linux: openSUSE 12.2, GNU Fortran compiler 4.7 (`gfortran`)
  - Windows: Windows 7, Visual Studio 2013, Intel Parallel Studio XE 2013 (`ifort.exe`)

Future work

- Apply this methodology to the most important GSLIB applications and utilities.
- Other GSLIB-based applications can also be benefited from this effort (following a similar acceleration methodology).
- If more processing power is needed, we can use **hardware acceleration devices** or **distributed computing**:
  - Early experiments using **Intel Xeon Phi (MICs)** co-processors (60 cores of execution).
  - **Nvidia CUDA-based GSLIB** which can use GPUs as back-end compute engines.
  - For really large computations, we are planning to implement an **MPI-based GSLIB** which can run in a cluster of distributed general-purpose computers.

Thanks for your attention!

Check for new updates and the latest release in:

`http://gslib.alges.cl`

Contact: `operedo [at] alges.cl`