Efficient R Parallel Loops on Long-Latency Platforms

Norm Matloff
University of California at Davis

Interface 2012
Rice University, May, 2012

Norm Matloff University of California at Davis

The Basic Problem

The Basic Problem

Given a loop of independent tasks,

```
parallel for i = 1,2,...,n
    do task i
```

The Basic Problem

Given a loop of independent tasks,

```
parallel for i = 1,2,...,n
    do task i
```

how to make this fast in R?

Example: Kendall's τ Correlation

Example: Kendall's τ Correlation

$$\hat{\tau} = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1_{((X_i, Y_i) \text{concord. with} (X_j, Y_j)}$$

Norm Matloff University of California at Davis

Example: Kendall's τ Correlation

$$\hat{\tau} = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1_{((X_i,Y_i) \text{concord. with} (X_j,Y_j)}$$

```
parallel for i = 1,2,...,n-1
  // here is task i:
  count = 0
  (nonparallel) for j = i+1,...,n
    count = count +
      1[((X[i],Y[i]) concord. with (X[j],Y[j])]
```

Example: Kendall's τ Correlation

$$\hat{\tau} = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1_{((X_i, Y_i) \text{concord. with} (X_j, Y_j)}$$

```
parallel for i = 1,2,...,n-1
  // here is task i:
  count = 0
  (nonparallel) for j = i+1,...,n
    count = count +
      1[((X[i],Y[i]) concord. with (X[j],Y[j])]
```

Major point: time(task i) $\searrow$ in i, thus issue of load balancing.

Example: All Possible Regressions

Example: All Possible Regressions

- Have n obs. on p vars.

Example: All Possible Regressions

- Have n obs. on p vars.
- Find "best" predictor set accord to some criterion, e.g. adjusted R^2.

Example: All Possible Regressions

- Have n obs. on p vars.
- Find "best" predictor set accord to some criterion, e.g. adjusted R^2.
- Evaluate criterion on all predictor sets of size $\leq$ some k.

Example: All Possible Regressions

- Have n obs. on p vars.
- Find "best" predictor set accord to some criterion, e.g. adjusted R^2.
- Evaluate criterion on all predictor sets of size $\leq$ some k.

```
parallel for i = 1,2,...,tot. # of models
   do regression i
```

Example: All Possible Regressions

- Have n obs. on p vars.
- Find "best" predictor set accord to some criterion, e.g. adjusted R^2.
- Evaluate criterion on all predictor sets of size $\leq$ some k.

```
parallel for i = 1,2,...,tot. # of models
   do regression i
```

Here time(task i) $\nearrow$ in i.

Goals of This Talk

Goals of This Talk

- Overview of classical shared-memory loop scheduling methods.

Goals of This Talk

- Overview of classical shared-memory loop scheduling methods.
- Discussion of how well these might adapt to parallel R.

Goals of This Talk

- Overview of classical shared-memory loop scheduling methods.
- Discussion of how well these might adapt to parallel R.
- Proposal of a new loop scheduling method, shown "optimal."

Goals of This Talk

- Overview of classical shared-memory loop scheduling methods.
- Discussion of how well these might adapt to parallel R.
- Proposal of a new loop scheduling method, shown "optimal."
- Case study (all possible regressions).

Goals of This Talk

- Overview of classical shared-memory loop scheduling methods.
- Discussion of how well these might adapt to parallel R.
- Proposal of a new loop scheduling method, shown "optimal."
- Case study (all possible regressions).
- Discussion of a possible algorithmic shortcut.

Research Literature

Research Literature

- Very extensively studied, e.g. (Hagerup, 1997).

Research Literature

- Very extensively studied, e.g. (Hagerup, 1997).
- However, most are for shared-memory machines, in which the overhead (task queue access latency) is low.

Research Literature

- Very extensively studied, e.g. (Hagerup, 1997).
- However, most are for shared-memory machines, in which the overhead (task queue access latency) is low.
- Some work for the long-latency case, e.g. (Yang and Chang, 2011), but limited.

Overhead Issues with Parallel R

Overhead Issues with Parallel R

- **snow:** serializes/deserializes communications; often used on clusters, incurring network delay

Overhead Issues with Parallel R

- **snow:** serializes/deserializes communications; often used on clusters, incurring network delay
- **Rmpi:** more flexible than **snow**, but still has the above serialization and network problems

Overhead Issues with Parallel R

- **snow:** serializes/deserializes communications; often used on clusters, incurring network delay
- **Rmpi:** more flexible than **snow**, but still has the above serialization and network problems
- **mclapply/multicore:** each call involves new Unix process creation

Overhead Issues with Parallel R

- **snow:** serializes/deserializes communications; often used on clusters, incurring network delay
- **Rmpi:** more flexible than **snow**, but still has the above serialization and network problems
- **mclapply/multicore:** each call involves new Unix process creation
- **gputools:** each call involves a GPU kernel invocation, major overhead

Overhead Issues with Parallel R

- **snow:** serializes/deserializes communications; often used on clusters, incurring network delay
- **Rmpi:** more flexible than **snow**, but still has the above serialization and network problems
- **mclapply/multicore:** each call involves new Unix process creation
- **gputools:** each call involves a GPU kernel invocation, major overhead
- These can be especially problematic with iterative algorithms, overhead incurred at every iteration.

Overhead Issues with Parallel R

- **snow:** serializes/deserializes communications; often used on clusters, incurring network delay
- **Rmpi:** more flexible than **snow**, but still has the above serialization and network problems
- **mclapply/multicore:** each call involves new Unix process creation
- **gputools:** each call involves a GPU kernel invocation, major overhead
- These can be especially problematic with iterative algorithms, overhead incurred at every iteration.

Bottom line: R typically needs larger applications, compared to C, in order to yield a "win."

Taxonomy of Classical Loop Scheduling Parameters

Taxonomy of Classical Loop Scheduling Parameters

- static: iterations pre-assigned to processes

Taxonomy of Classical Loop Scheduling Parameters

- static: iterations pre-assigned to processes
- dynamic: task queue or equivalent

Taxonomy of Classical Loop Scheduling Parameters

- static: iterations pre-assigned to processes
- dynamic: task queue or equivalent
- chunk size: number of consecutive values of i handled by a process

Taxonomy of Classical Loop Scheduling Parameters

- static: iterations pre-assigned to processes
- dynamic: task queue or equivalent
- chunk size: number of consecutive values of i handled by a process
- above are options in the shared-memory system OpenMP

Tradeoffs

Tradeoffs

- static case:

Tradeoffs

- static case:
  - no task queue overhead, but

Tradeoffs

- static case:
  - no task queue overhead, but
  - possible load balance problem (idle processes near end).

Tradeoffs

- static case:
  - no task queue overhead, but
  - possible load balance problem (idle processes near end).
- dynamic case:

Tradeoffs

- static case:
  - no task queue overhead, but
  - possible load balance problem (idle processes near end).
- dynamic case:
  - larger chunk size $\Rightarrow$ smaller overhead but poorer load balance

Tradeoffs

- static case:
  - no task queue overhead, but
  - possible load balance problem (idle processes near end).
- dynamic case:
  - larger chunk size $\Rightarrow$ smaller overhead but poorer load balance
  - smaller chunk size $\Rightarrow$ larger overhead but better load balance

Tradeoffs

- static case:
  - no task queue overhead, but
  - possible load balance problem (idle processes near end).
- dynamic case:
  - larger chunk size $\Rightarrow$ smaller overhead but poorer load balance
  - smaller chunk size $\Rightarrow$ larger overhead but better load balance
- time-varying chunk size:

Tradeoffs

- static case:
  - no task queue overhead, but
  - possible load balance problem (idle processes near end).
- dynamic case:
  - larger chunk size $\Rightarrow$ smaller overhead but poorer load balance
  - smaller chunk size $\Rightarrow$ larger overhead but better load balance
- time-varying chunk size:
  - large for early i, smaller near the end;

Tradeoffs

- static case:
  - no task queue overhead, but
  - possible load balance problem (idle processes near end).
- dynamic case:
  - larger chunk size $\Rightarrow$ smaller overhead but poorer load balance
  - smaller chunk size $\Rightarrow$ larger overhead but better load balance
- time-varying chunk size:
  - large for early i, smaller near the end; aims for "best of both worlds"

Tradeoffs

- static case:
  - no task queue overhead, but
  - possible load balance problem (idle processes near end).
- dynamic case:
  - larger chunk size $\Rightarrow$ smaller overhead but poorer load balance
  - smaller chunk size $\Rightarrow$ larger overhead but better load balance
- time-varying chunk size:
  - large for early i, smaller near the end; aims for "best of both worlds"
  - **guided** option in OpenMP

A "New" Scheduling Method

A "New" Scheduling Method

Notation:

- **ni:** Total number of iterations in loop.
- **np:** Number of processes (e.g. num. workers in **snow**).

A "New" Scheduling Method

Notation:

- **ni:** Total number of iterations in loop.
- **np:** Number of processes (e.g. num. workers in **snow**).

Method:

- Randomly permute the i's , i.e. (1,2,...,**ni**);

A "New" Scheduling Method

Notation:

- **ni:** Total number of iterations in loop.
- **np:** Number of processes (e.g. num. workers in **snow**).

Method:

- Randomly permute the i's , i.e. (1,2,...,**ni**); use static,

A "New" Scheduling Method

Notation:

- **ni:** Total number of iterations in loop.
- **np:** Number of processes (e.g. num. workers in **snow**).

Method:

- Randomly permute the i's , i.e. (1,2,...,**ni**); use static, with full chunk size (**ni/np**).

A "New" Scheduling Method

Notation:

- **ni:** Total number of iterations in loop.
- **np:** Number of processes (e.g. num. workers in **snow**).

Method:

- Randomly permute the i's , i.e. (1,2,...,**ni**); use static, with full chunk size (**ni/np**).
- Sometimes mentioned briefly in lit., but "new," since not studied analytically before.

A "New" Scheduling Method

Notation:

- **ni:** Total number of iterations in loop.
- **np:** Number of processes (e.g. num. workers in **snow**).

Method:

- Randomly permute the i's , i.e. (1,2,...,**ni**); use static, with full chunk size (**ni/np**).
- Sometimes mentioned briefly in lit., but "new," since not studied analytically before.
- Easy to show this method asymp. yields full load balance.

A "New" Scheduling Method

Notation:

- **ni:** Total number of iterations in loop.
- **np:** Number of processes (e.g. num. workers in **snow**).

Method:

- Randomly permute the i's , i.e. (1,2,...,**ni**); use static, with full chunk size (**ni/np**).
- Sometimes mentioned briefly in lit., but "new," since not studied analytically before.
- Easy to show this method asymp. yields full load balance.
- Has zero overhead, achieves full load balance $\Rightarrow$ optimal!

A "New" Scheduling Method

Notation:

- **ni:** Total number of iterations in loop.
- **np:** Number of processes (e.g. num. workers in **snow**).

Method:

- Randomly permute the i's , i.e. (1,2,...,**ni**); use static, with full chunk size (**ni/np**).
- Sometimes mentioned briefly in lit., but "new," since not studied analytically before.
- Easy to show this method asymp. yields full load balance.
- Has zero overhead, achieves full load balance $\Rightarrow$ optimal!
- But only asympotically. :-)

A "New" Scheduling Method

Notation:

- **ni:** Total number of iterations in loop.
- **np:** Number of processes (e.g. num. workers in **snow**).

Method:

- Randomly permute the i's , i.e. (1,2,...,**ni**); use static, with full chunk size (**ni/np**).
- Sometimes mentioned briefly in lit., but "new," since not studied analytically before.
- Easy to show this method asymp. yields full load balance.
- Has zero overhead, achieves full load balance $\Rightarrow$ optimal!
- But only asympotically. :-)
- Not a bad choice, if you don't want to bother tweaking chunk size, etc.

A "New" Scheduling Method

Notation:

- **ni:** Total number of iterations in loop.
- **np:** Number of processes (e.g. num. workers in **snow**).

Method:

- Randomly permute the i's , i.e. (1,2,...,**ni**); use static, with full chunk size (**ni/np**).
- Sometimes mentioned briefly in lit., but "new," since not studied analytically before.
- Easy to show this method asymp. yields full load balance.
- Has zero overhead, achieves full load balance $\Rightarrow$ optimal!
- But only asympotically. :-)
- Not a bad choice, if you don't want to bother tweaking chunk size, etc. Simplify your life!

Proof of Load Balance

Proof of Load Balance

- No assumptions (contrast to other research); data not even considered random.

Proof of Load Balance

- No assumptions (contrast to other research); data not even considered random.
- Chunk size c is **ni** / **np**.

Proof of Load Balance

- No assumptions (contrast to other research); data not even considered random.
- Chunk size c is **ni** / **np**.
- Set t_j = task time for iter. j; set μ and σ^2 to mean and variance of $t_1,...,t_{ni}$.

Proof of Load Balance

- No assumptions (contrast to other research); data not even considered random.
- Chunk size c is **ni** / **np**.
- Set t_j = task time for iter. j; set μ and σ^2 to mean and variance of $t_1,...,t_{ni}$.
- Cast the problem as one of sampling without replacement.

Proof of Load Balance

- No assumptions (contrast to other research); data not even considered random.
- Chunk size c is **ni** / **np**.
- Set t_j = task time for iter. j; set μ and σ^2 to mean and variance of $t_1,...,t_{ni}$.
- Cast the problem as one of sampling without replacement.
- Total time for iters. for process j has coeff. of variation

$$\frac{\sqrt{(1-\frac{c}{ni})c\sigma^2}}{c\mu} \rightarrow 0 \text{ as } c \rightarrow \infty$$

- Etc.

Proof of Load Balance

- No assumptions (contrast to other research); data not even considered random.
- Chunk size c is **ni** / **np**.
- Set t_j = task time for iter. j; set μ and σ^2 to mean and variance of $t_1,...,t_{ni}$.
- Cast the problem as one of sampling without replacement.
- Total time for iters. for process j has coeff. of variation

$$\frac{\sqrt{(1-\frac{c}{ni})c\sigma^2}}{c\mu} \to 0 \text{ as } c \to \infty$$

- Etc.
- So, total task time $\approx$ constant across processes, i.e. have load balance.

Scheduling Options in Snow

Scheduling Options in Snow

Our analysis here will focus on **snow**.

Scheduling Options in Snow

Our analysis here will focus on **snow**. Scheduling options:

Scheduling Options in Snow

Our analysis here will focus on **snow**. Scheduling options:

- **clusterApply():** static

Scheduling Options in Snow

Our analysis here will focus on **snow**. Scheduling options:

- **clusterApply():** static
- **clusterApplyLB():** dynamic

Scheduling Options in Snow

Our analysis here will focus on **snow**. Scheduling options:

- **clusterApply():** static
- **clusterApplyLB():** dynamic
- both limited to a fixed chunk size of 1

Scheduling Options in Snow

Our analysis here will focus on **snow**. Scheduling options:

- **clusterApply():** static
- **clusterApplyLB():** dynamic
- both limited to a fixed chunk size of 1
- chunk size > 1 must be programmed with user's own code

Code for All Possible Regressions

Code for All Possible Regressions

```
1  prsnow <- function(cls,x,y,k,
2        rnd=F,chunk=NULL,dyn=F) {
3     p <- ncol(x); allc <<- genallcombs(p,k)
4     if (rnd) allc <- randperm(allc)
5     ni <<- nrow(allc); np <- length(cls))
6     if (is.null(chunk)) chunk <- floor(ni/np))
7     chunk <<- chunk
8     clusterExport(cls,c("allc","ni","chunk","x"
9     clusterExport(cls,"do1pset")
10    is <- seq(1,ni,chunk)
11    if (!dyn) { ar2s <<-
12       clusterApply(cls,is,dochunk)
13    } else { ar2s <<-
14       clusterApplyLB(cls,is,dochunk)
15    }
16 }
```

Code, cont'd.

Code, cont'd.

```
1  dochunk <- function(psetchunk) {
2      lasttask <- min(psetchunk+chunk-1,nc)
3      out <- NULL
4      for (tasknum in psetchunk:lasttask) {
5          out <- c(out,do1pset(tasknum))
6      }
7      return(out)
8  }
9
10 do1pset <- function(pset) {
11     onerow <- allcombs[pset,]
12     nps  <- onerow[1]
13     ps <- onerow[2:(1+nps)]
14     slm <- summary(lm(y ~ x[,ps]))
15     return(Reduce(paste,c(slm$adj.
16         r.squared,myinfo$id,onerow[-1])))
17 }
```

Options

Options

- **chunk:** Chunk size. Default value is **ni**/**np**.

Options

- **chunk:** Chunk size. Default value is **ni**/**np**.
- **dyn:** Use dynamic scheduling, i.e. **clusterApplyLB()** instead of **clusterApply()**. Default value is False.

Options

- **chunk:** Chunk size. Default value is **ni**/**np**.
- **dyn:** Use dynamic scheduling, i.e. **clusterApplyLB()** instead of **clusterApply()**. Default value is False.
- **rnd:** Use random scheduling. Default value is False.

Timings

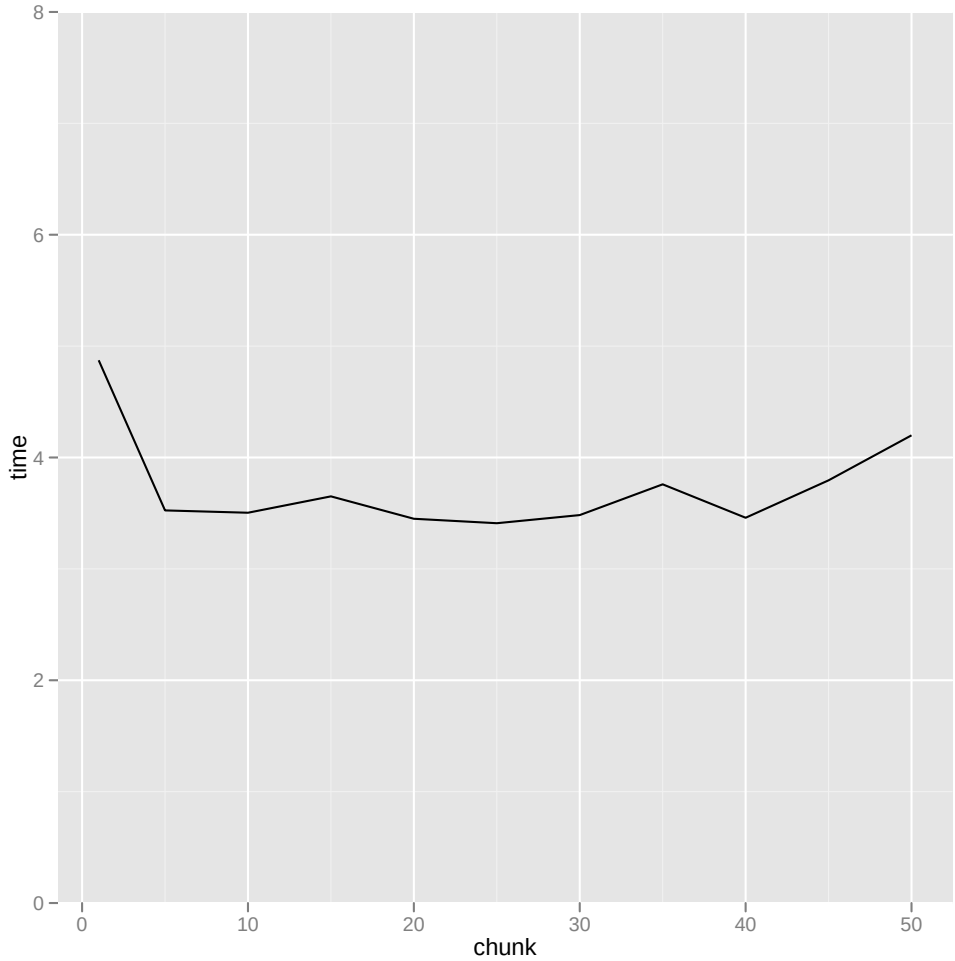
Timings

- 10,000 obs., 8 predictors
- k = 4 (i.e. up to 4 preds.)
- 2 procs., same machine
- chunk sizes 1,5,10,...,50; 5 reps. each

Norm Matloff University of California at Davis

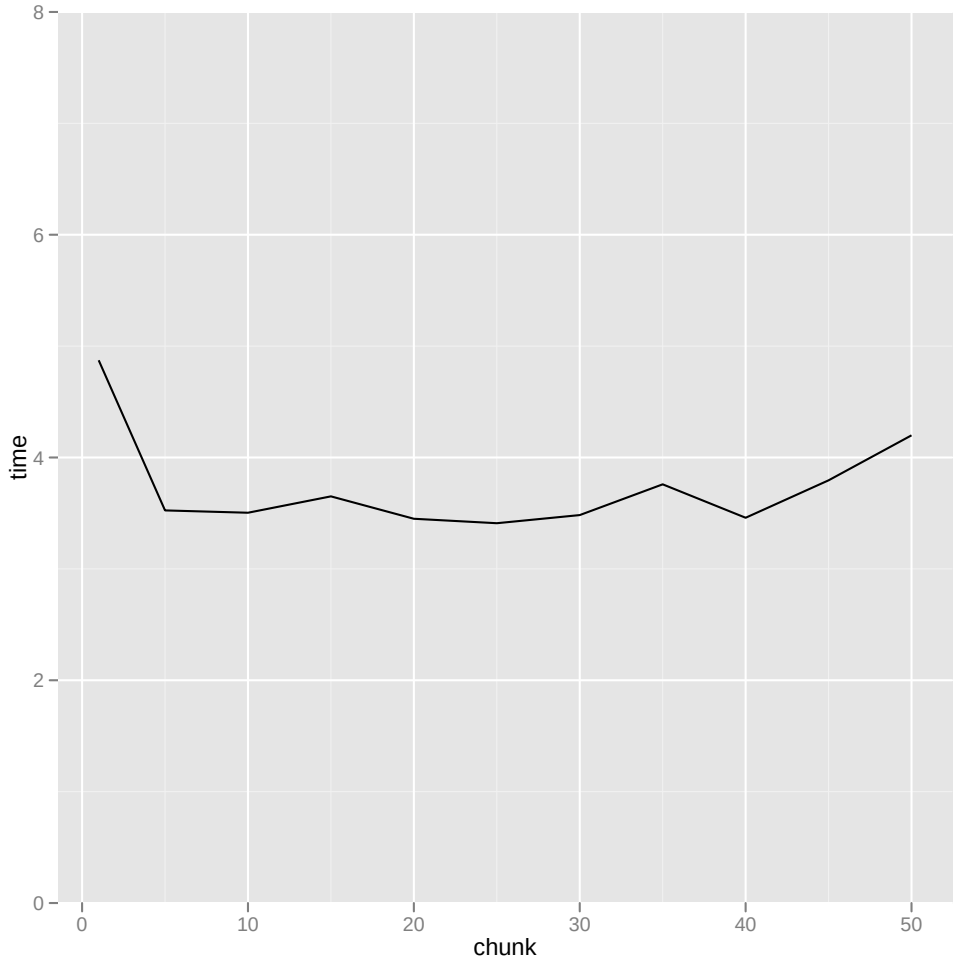
Timings

- 10,000 obs., 8 predictors
- k = 4 (i.e. up to 4 preds.)
- 2 procs., same machine
- chunk sizes 1,5,10,...,50; 5 reps. each



Timings

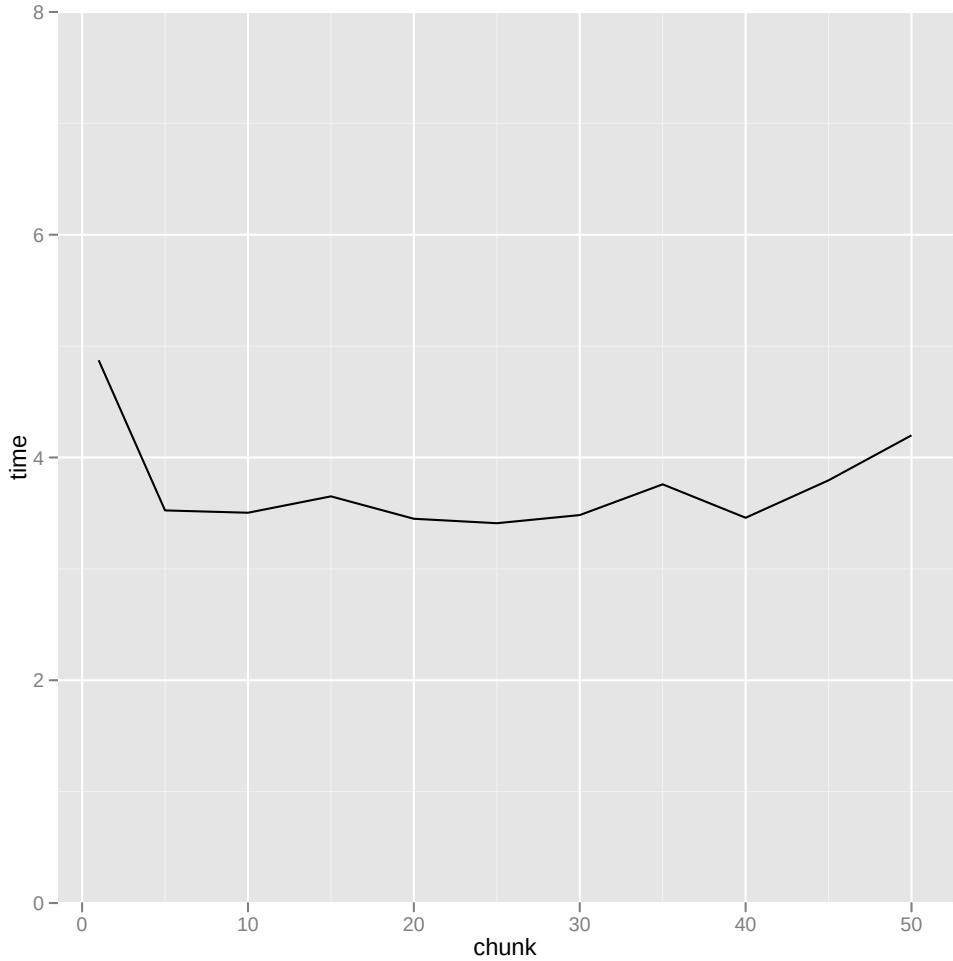
- 10,000 obs., 8 predictors
- k = 4 (i.e. up to 4 preds.)
- 2 procs., same machine
- chunk sizes 1,5,10,...,50; 5 reps. each



Chunks too small $\Rightarrow$ overhead problem.

Timings

- 10,000 obs., 8 predictors
- k = 4 (i.e. up to 4 preds.)
- 2 procs., same machine
- chunk sizes 1,5,10,...,50; 5 reps. each



Chunks too small ⇒ overhead problem.
Chunks too large ⇒ load balance problem.

Network Platform

Network Platform

Same setting, but on a network platform.

Network Platform

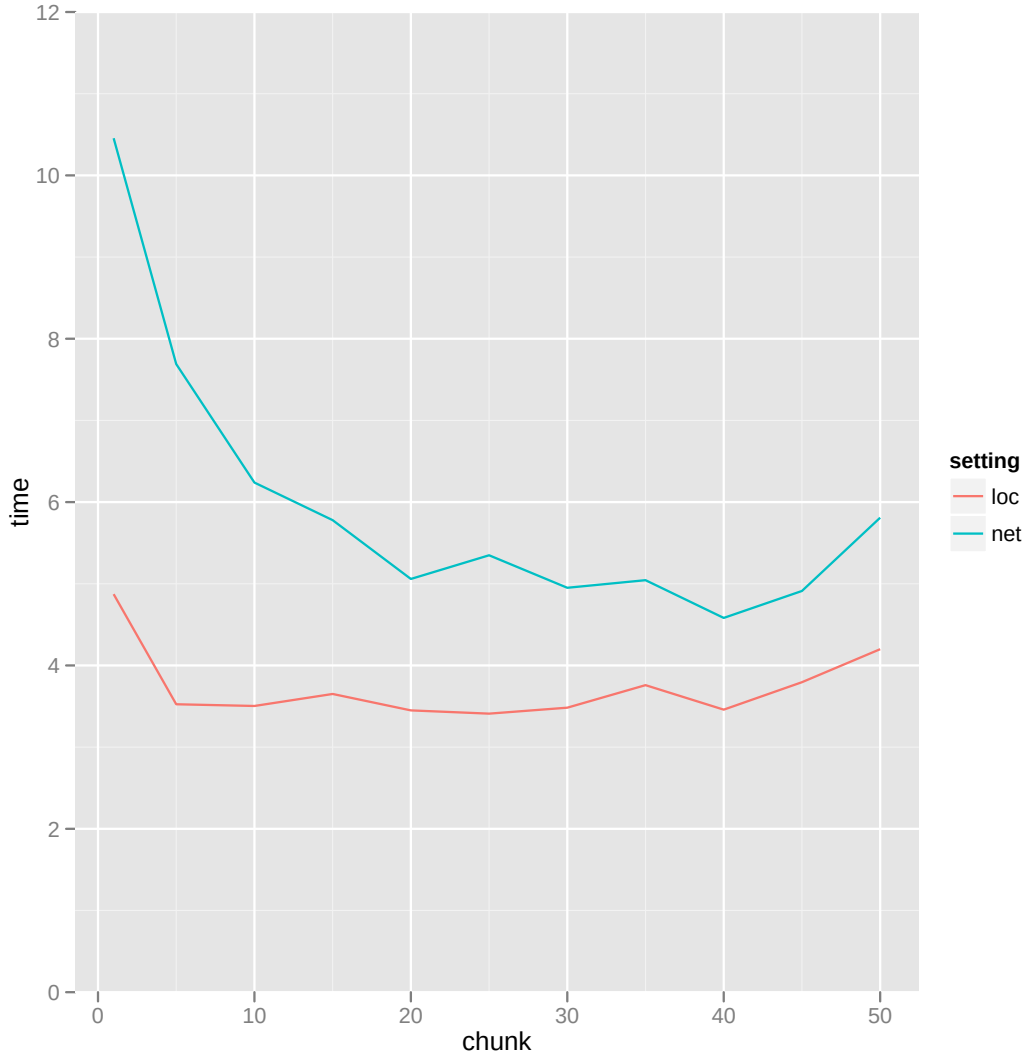
Same setting, but on a network platform.

Worker nodes chosen to be distant from manager node, to highlight overhead issue.

Network Platform

Same setting, but on a network platform.

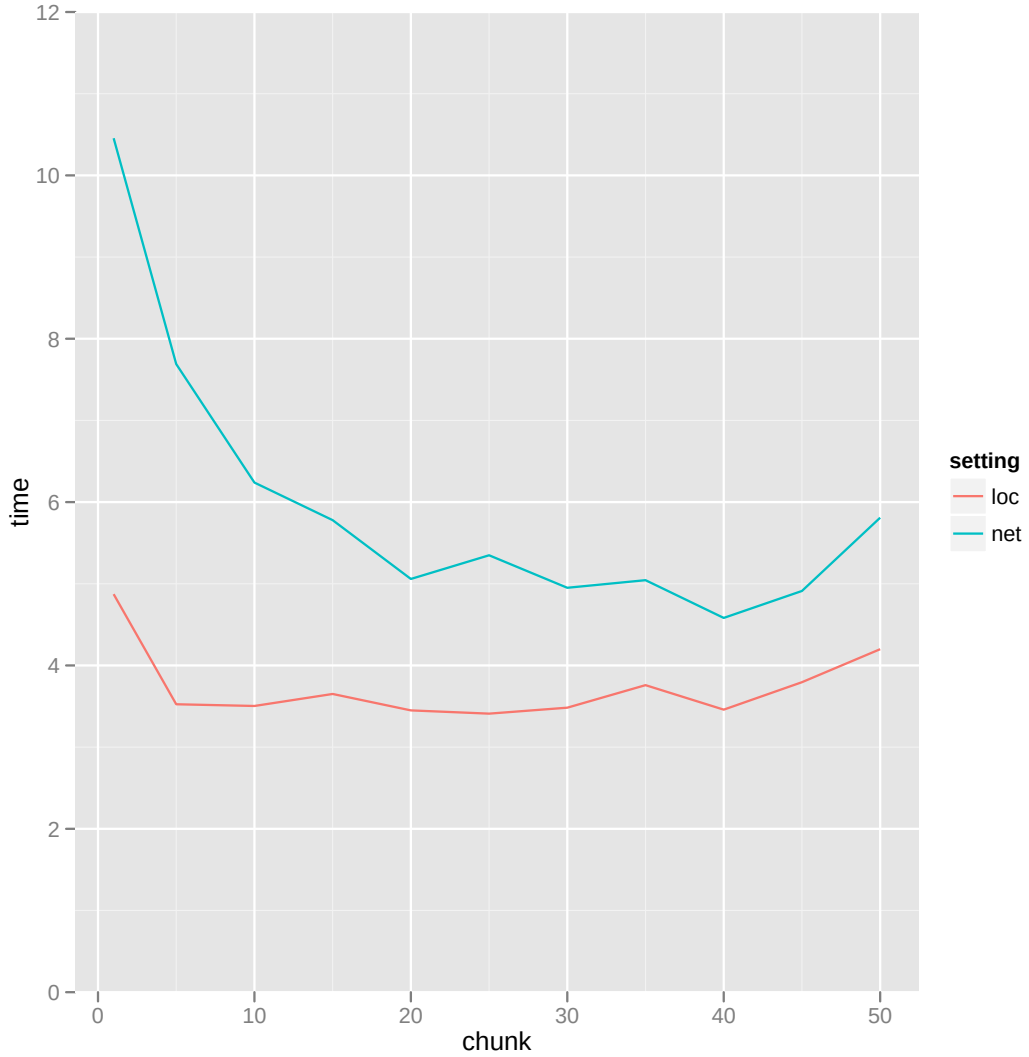
Worker nodes chosen to be distant from manager node, to highlight overhead issue.



Network Platform

Same setting, but on a network platform.

Worker nodes chosen to be distant from manager node, to highlight overhead issue.



Impact of choice of chunk size more dramatic here.

Comparison to Random Scheduling

Comparison to Random Scheduling

setting	best chunk	worst chunk	random
localhost	3.410	4.873	3.794
network	4.582	10.455	4.723

Comparison to Random Scheduling

setting	best chunk	worst chunk	random
localhost	3.410	4.873	3.794
network	4.582	10.455	4.723

Again, random method only asymp. optimal, but good choice if don't want to spend time tweaking the chunk size.

Scalability

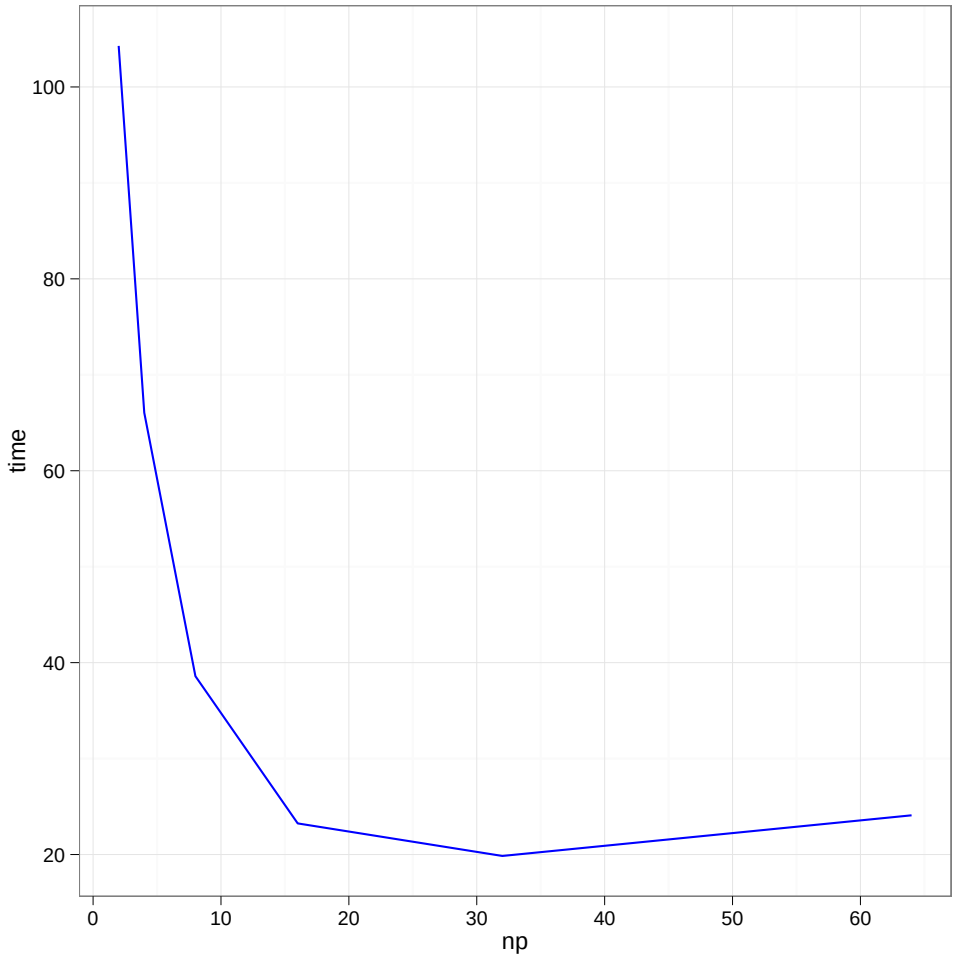
Scalability

- 10000 obs., 20 vars.
- **np** = 2,4,8,16,32,64, on localhost (> 64 cores)
- Random sched. ("representative").

Norm Matloff University of California at Davis

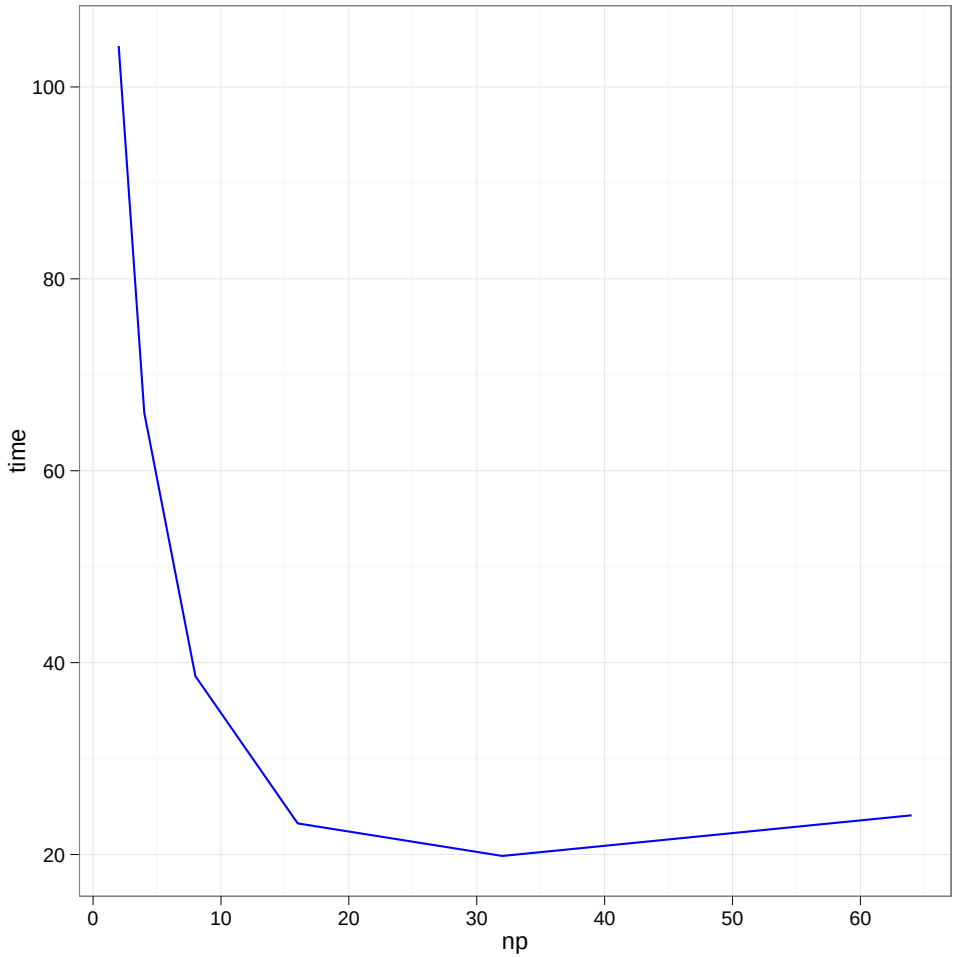
Scalability

- 10000 obs., 20 vars.
- **np** = 2,4,8,16,32,64, on localhost (> 64 cores)
- Random sched. ("representative").



Scalability

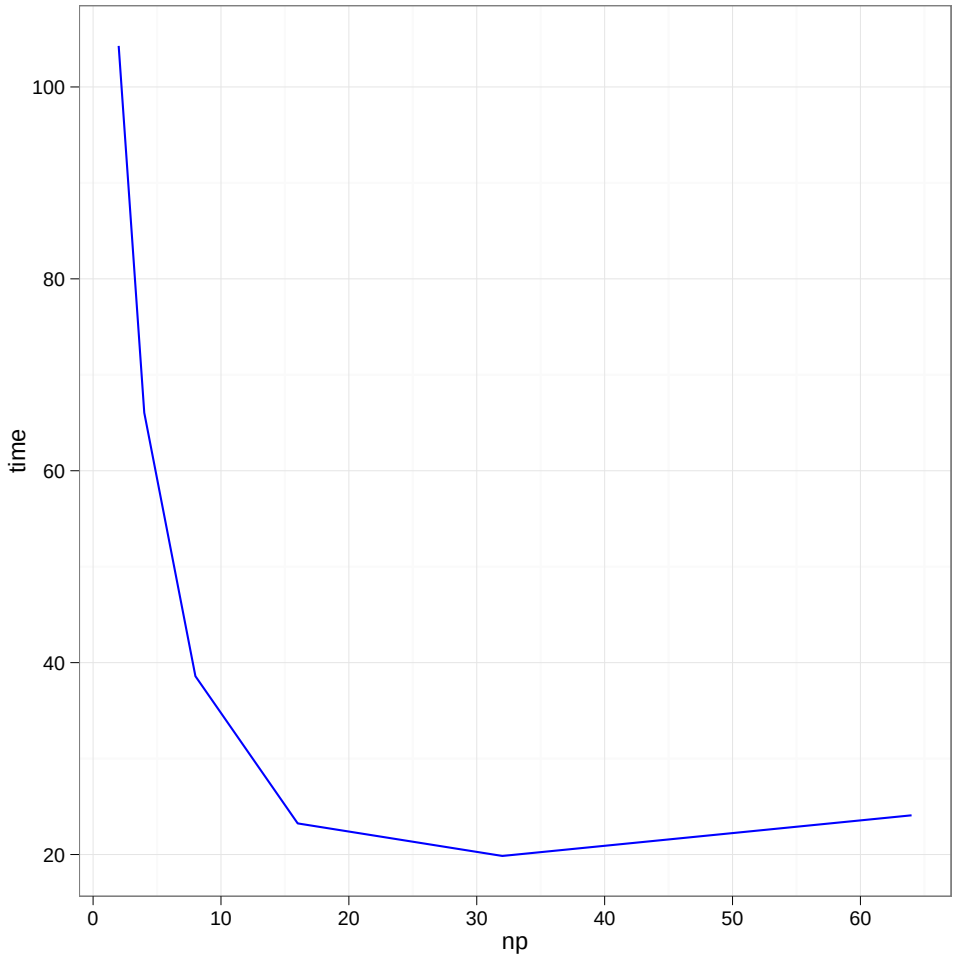
- 10000 obs., 20 vars.
- **np** = 2,4,8,16,32,64, on localhost (> 64 cores)
- Random sched. ("representative").



Overhead ⇒ diminishing returns

Scalability

- 10000 obs., 20 vars.
- **np** = 2,4,8,16,32,64, on localhost (> 64 cores)
- Random sched. ("representative").



Overhead ⇒ diminishing returns—eventually negative.

Algorithmic Speedup

Algorithmic Speedup

- Exploit matrix update:

Algorithmic Speedup

- Exploit matrix update: Get new $(X'X)^{-1}$ from the old one when add a new variable.

Algorithmic Speedup

- Exploit matrix update: Get new $(X'X)^{-1}$ from the old one when add a new variable. Possibly get a speedup?

Algorithmic Speedup

- Exploit matrix update: Get new $(X'X)^{-1}$ from the old one when add a new variable. Possibly get a speedup?
- Scheduling may be rather intricate.

Efficient R Parallel Loops on Long-Latency Platforms

Norm Matloff
University of California at Davis

Norm Matloff University of California at Davis

Slides available at *http://heather.cs.ucdavis.edu/RiceSlides.pdf*.

To learn about parallel programming, see my open source book at `http://heather.cs.ucdavis.edu/parprocbook`.