# CS 596:Introduction to Parallel Computing
# Topic: Parallel Performance

Mary Thomas

Department of Computer Science
Computational Science Research Center (CSRC)
San Diego State University (SDSU)

Posted: 02/15/17
Last Update: 02/15/17

## Table of Contents

## Timing Serial or Parallel Code

**What/how to measure?**

- CPU_time? System? Hardware? I/O? Human?
- What is start/stop time, how to compute?
- Where to time? Critical blocks?
- Subprograms? Overhead?
- Difference between $T_{wall}$, $T_{cpu}$, $T_{user}$
- Data type: integer, char, float, double...

**Units/Metrics?**

- Time: seconds, milliseconds, micro, nano
- Frequency: Hz ($1/sec$)
- Scale: Kilo, Mega, Giga, Tera, Peta, .
- Operation counts:
    - FLOPS: floating point operations per second

**In general, performance is measured not calculated**

## Total Program Time

Total computer program time is a function of a large number of variables: computer hardware (cpu, memory, software, network), and the program being run (algorithm, problem size, # Tasks, complexity)

$$T = \mathcal{F}\left(ProbSize,\ Tasks,\ I/O,\ \ldots\right)$$

Source: `http://en.wikipedia.org/wiki/Wall-clock_time`

## Where to time the code?

- Look for where the most work is being done.
- You don't need to time all of the program
- Critical Blocks:
  - Points in the code where you expect to do a large amount of work
  - Problem size dependencies
  - 2D matrix: $\vartheta\,(n * m)$, Binary Search Tree: $\vartheta\,(\log n)$
- Input and Output statements:
  - STDIO/STDIN
  - File I/O

## Wallclock Time: $T_{wall}$

A measure of the real time that elapses from the start to the end of a computer program.

It is the difference between the time at which the program finishes and the time at which the program started.

$$T_{wall} = T_{CPU} + T_{I/O} + T_{Idle} + T_{other}$$

Source: `http://en.wikipedia.org/wiki/Wall-clock_time`

## Wallclock Time: $T_{wall}$

$$T_{wall} = T_{CPU} + T_{I/O} + T_{Idle} + T_{other}$$

- $T_{Wall}$: The total (or real) time that has elapsed from the start to the completion of a computer program or task.
- $T_{CPU}$: The amount of time for which a central processing unit (CPU) is used for processing instructions of a computer program or operating system.
- $T_{I/O}$: The time spent by a computer program reading/writing data to/from files such as /STDIN/STDERR, local data files, remote data services or databases.
- $T_{Idle}$: The time spent by a computer program waiting for execution instructions.
- $T_{overhead}$: The amount of time required to set up a computer program including setting up hardware, local and remote data and resources, network connections, messages.

## Total Parallel Program Time

- The total parallel program run time is a function of a large number of variables: number of processing elements (PEs); communication; hardware (cpu, memory, software, network), and the program being run (algorithm, problem size, # Tasks, complexity, data distribution); parallel libraries:

$$T = \mathcal{F}\left(PEs, N,\ Tasks,\ I/O,\ Communication, \ldots\right)$$

- The execution time required to run a problem of size N on processor $i$, is a function of the time spent in different parts of the program (computation, communication, I/O, idle):

$$T^i = T^i_{comp} + T^i_{comm} + T^i_{io} + T^i_{idle}$$

- The total time is the sum of the times over all processes averaged over the number of the processors: $T =$

$$\frac{1}{p}\left(\sum_{i=0}^{p-1} T_{comp} + \sum_{i=0}^{p-1} T_{comm} + \sum_{i=0}^{p-1} T_{io} + \sum_{i=0}^{p-1} T_{idel}\right)$$

## Speedup

- Refers to how much faster the parallel algorithm runs than a corresponding sequential algorithm (non-MPI).
- $T_{ser}$ = time between when *serial* program begins to when it completes its tasks.
- $T_{par}$ = time between when *first* processor begins execution to when the *last* processor completes its tasks.
- The Speedup is defined to be: $S_p = \frac{T_{ser}}{T_{par}}$
- Where:
  - $p \equiv$ number of cores (processors, PE's)
  - $T_{ser} \equiv$ serial execution time
  - $T_{par} \equiv$ parallel execution time
- Linear speedup, or ideal speedup, is obtained when $S_p = p$, or

$$T_{par} = T_{ser} \,/\, p$$

## Efficiency

- Estimation of how well the processors are used to solve the problem vs. effort is wasted in communication and synchronization.
- $T_{elap} ==$ time between when first processor begins execution to when the last processor completes its tasks

$$E = \frac{S}{p} = \frac{\left(\frac{T_{serial}}{T_{parallel}}\right)}{p} = \frac{T_{serial}}{p \cdot T_{parallel}}$$

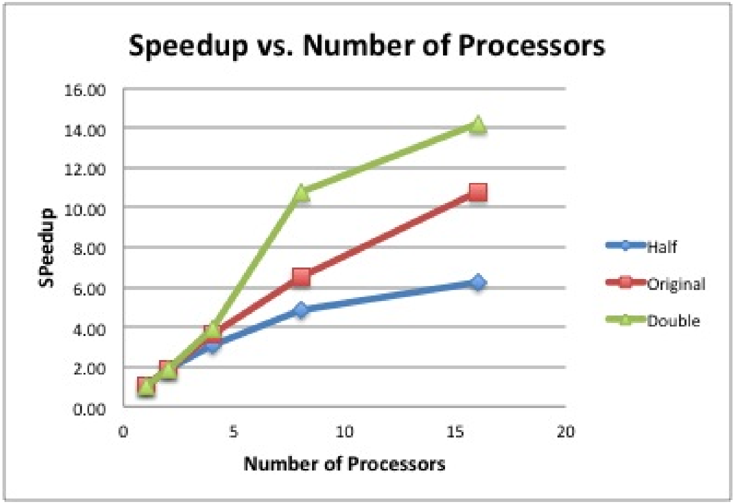
- Where:
  - $p ==$ number of cores (processors, PE's)
  - $T_{ser} ==$ serial execution time
  - $T_{par} ==$ parallel execution time
- Efficiency is ypically between *zero* and *one*

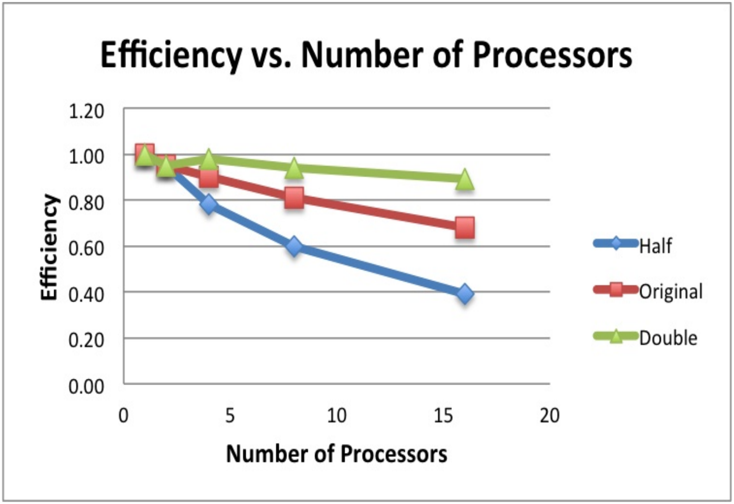
| RunTimes | p | 1 | 2 | 4 | 8 | 16 |
|---|---|---|---|---|---|---|
| Half | RT | 1.00 | 0.53 | 0.32 | 0.21 | 0.16 |
| Original | RT | 1.00 | 0.53 | 0.28 | 0.15 | 0.09 |
| Double | RT | 1.00 | 0.53 | 0.26 | 0.09 | 0.07 |

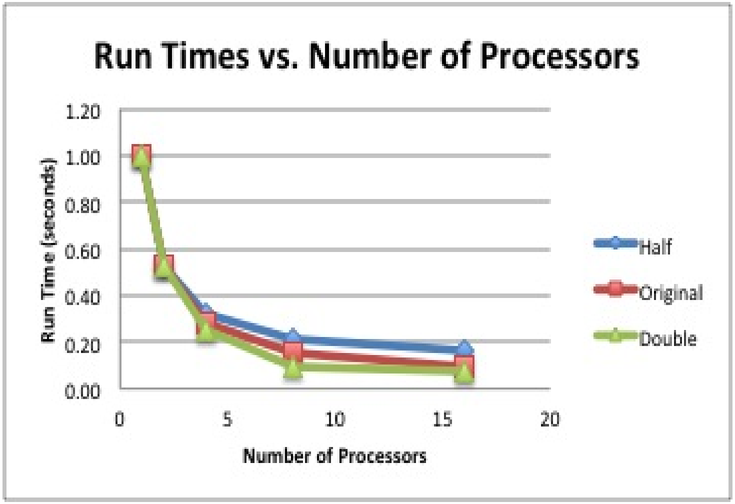
| ProbSize | p | 1 | 2 | 4 | 8 | 16 |
|---|---|---|---|---|---|---|
| Half | S | 1.00 | 1.90 | 3.10 | 4.80 | 6.20 |
|  | E | 1.00 | 0.95 | 0.78 | 0.60 | 0.39 |
| Original | S | 1.00 | 1.90 | 3.60 | 6.50 | 10.80 |
|  | E | 1.00 | 0.95 | 0.90 | 0.81 | 0.68 |
| Double | S | 1.00 | 1.90 | 3.90 | 10.80 | 14.20 |
|  | E | 1.00 | 0.95 | 0.98 | 0.94 | 0.89 |

Test data from showing the effect of problem size on the run times (RT), speedup (S) and efficiency (E).

Source: Pacheco IPP (Ch 2)


Speedup vs. Number of Processors
SPeedup
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
0
5
10
15
20
Number of Processors
Half
Original
Double


Efficiency vs. Number of Processors
Efficiency
1.20
1.00
0.80
0.60
0.40
0.20
0.00
0
5
10
15
20
Number of Processors
Half
Original
Double


Run Times vs. Number of Processors
Run Time (seconds)
1.20
1.00
0.80
0.60
0.40
0.20
0.00
0
5
10
15
20
Number of Processors
Half
Original
Double

## Effect of Overhead

- Overhead is associated with work done by program and system on non-computational activities
- These include process management, backend communications, page swapping and data access control, security, etc.

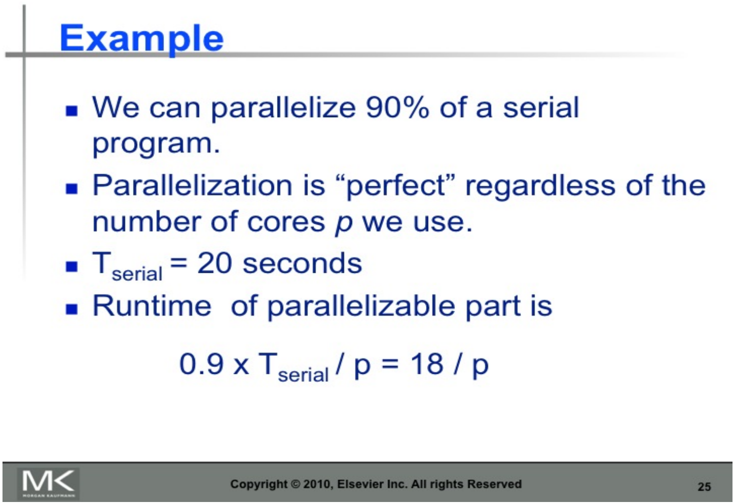
$$\boxed{T_{par} = \frac{T_{ser}}{p} + T_{overhead}}$$

## Amdahl's Law

- Used to find the maximum expected improvement to an overall system when only part of the system is improved.
- Often used in parallel computing to predict the theoretical maximum speedup using multiple processors.

**Definition:** If $B$ is the fraction of the algorithm that is strictly serial, and $p$ is the number of processes (cores, threads, etc.), then the time $T(n)$ required for a program to execute can be written as:

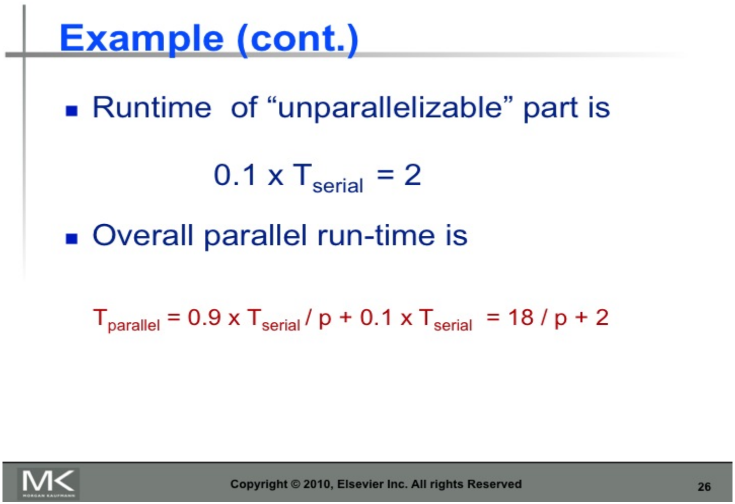
$$\begin{aligned} T(n) &= T_{ser} + T_{par} \\ &= T(1)\,B + \frac{T(1)}{n}(1-B) \\ &= T(1)\left(B + \frac{1}{n}(1-B)\right) \end{aligned}$$



## Example

- We can parallelize 90% of a serial program.
- Parallelization is "perfect" regardless of the number of cores *p* we use.
- $T_{serial} = 20$ seconds
- Runtime of parallelizable part is

$$0.9 \times T_{serial} / p = 18 / p$$



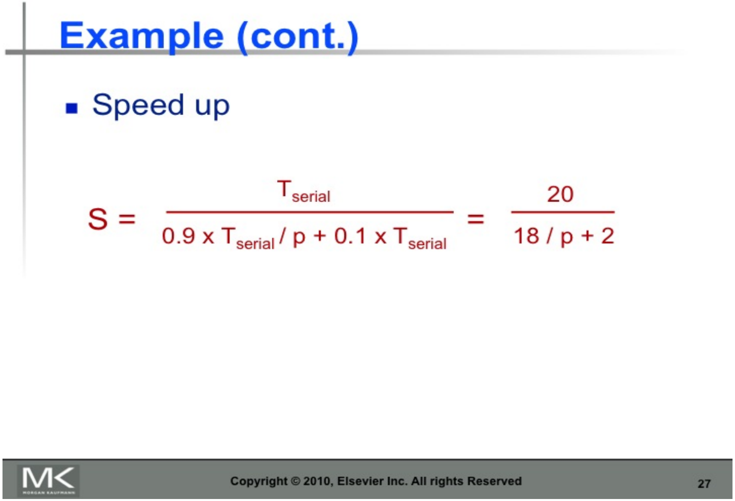
## Example (cont.)

- Runtime of "unparallelizable" part is

$$0.1 \times T_{serial} = 2$$

- Overall parallel run-time is

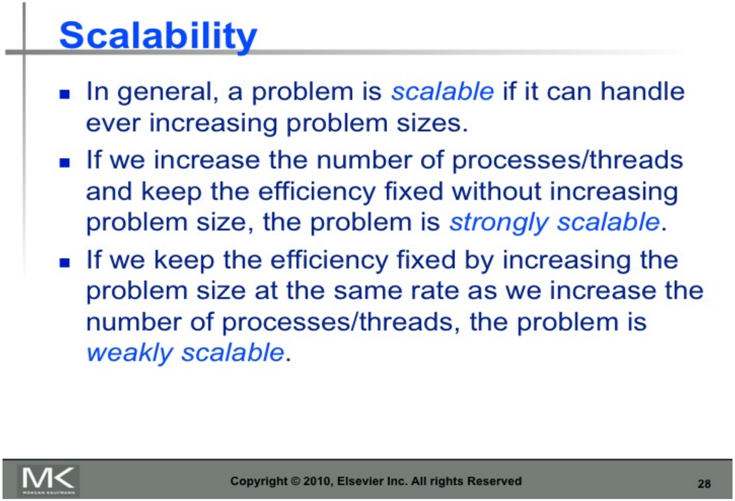
$$T_{parallel} = 0.9 \times T_{serial} / p + 0.1 \times T_{serial} = 18 / p + 2$$



## Example (cont.)

- Speed up

$$S = \frac{T_{serial}}{0.9 \times T_{serial} / p + 0.1 \times T_{serial}} = \frac{20}{18 / p + 2}$$



## Scalability

- In general, a problem is *scalable* if it can handle ever increasing problem sizes.
- If we increase the number of processes/threads and keep the efficiency fixed without increasing problem size, the problem is *strongly scalable*.
- If we keep the efficiency fixed by increasing the problem size at the same rate as we increase the number of processes/threads, the problem is *weakly scalable*.

## Customized Timings: Parallel Framework

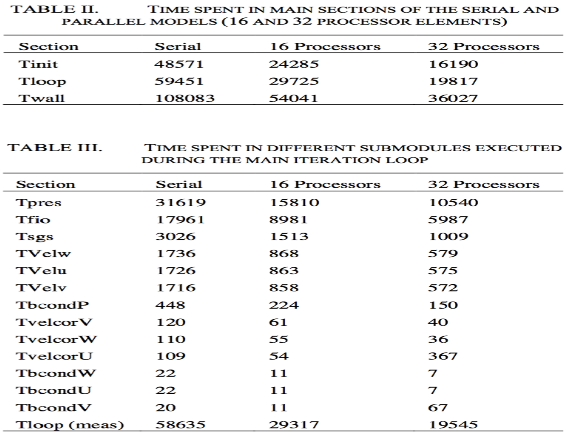
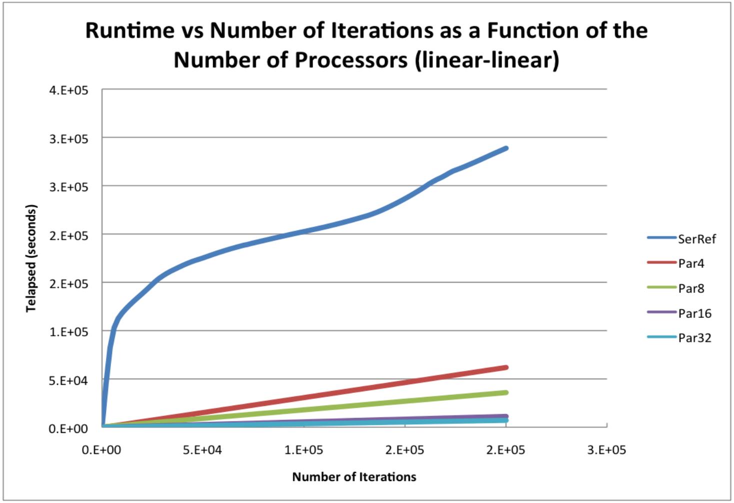


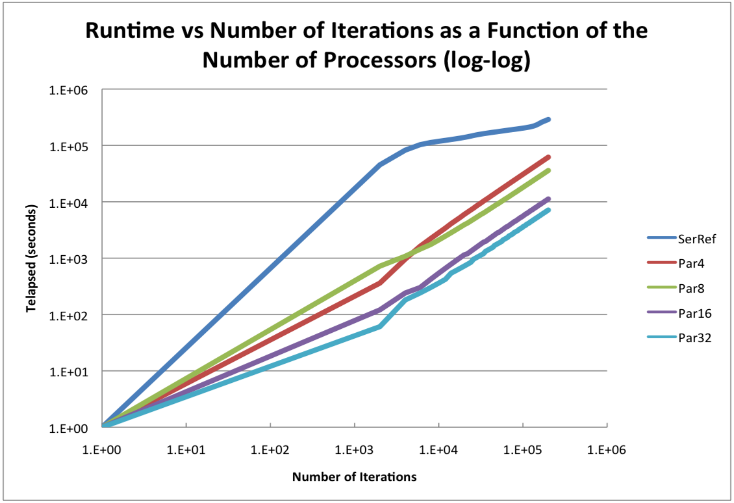
TABLE II. TIME SPENT IN MAIN SECTIONS OF THE SERIAL AND PARALLEL MODELS (16 AND 32 PROCESSOR ELEMENTS)

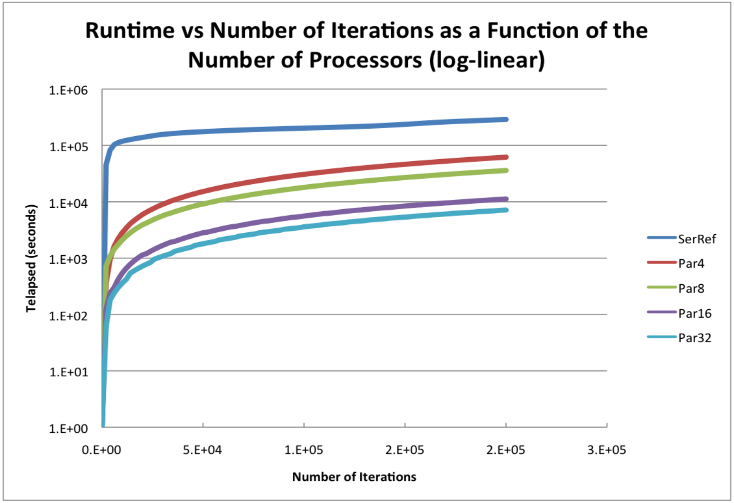
| Section | Serial | 16 Processors | 32 Processors |
|---|---|---|---|
| Tinit | 48571 | 24285 | 16190 |
| Tloop | 59451 | 29725 | 19817 |
| Twall | 108083 | 54041 | 36027 |

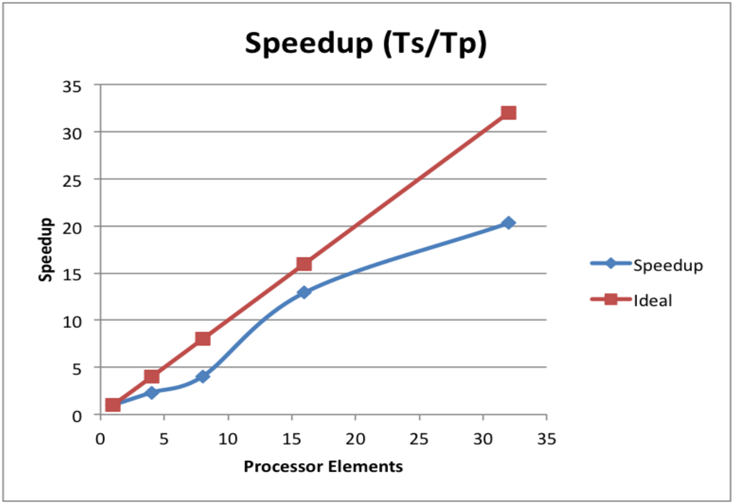
TABLE III. TIME SPENT IN DIFFERENT SUBMODULES EXECUTED DURING THE MAIN ITERATION LOOP

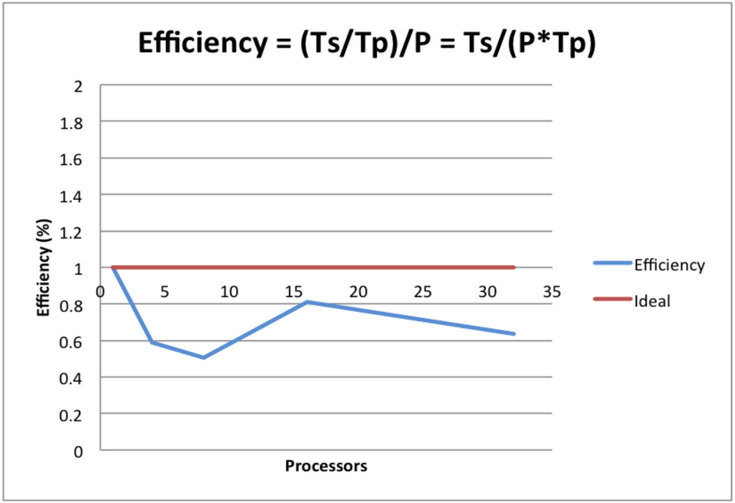
| Section | Serial | 16 Processors | 32 Processors |
|---|---|---|---|
| Tpres | 31619 | 15810 | 10540 |
| Tfio | 17961 | 8981 | 5987 |
| Tsgs | 3026 | 1513 | 1009 |
| TVelw | 1736 | 868 | 579 |
| TVelu | 1726 | 863 | 575 |
| TVelv | 1716 | 858 | 572 |
| TbcondP | 448 | 224 | 150 |
| TvelcorV | 120 | 61 | 40 |
| TvelcorW | 110 | 55 | 36 |
| TvelcorU | 109 | 54 | 367 |
| TbcondW | 22 | 11 | 7 |
| TbcondU | 22 | 11 | 7 |
| TbcondV | 20 | 11 | 67 |
| Tloop (meas) | 58635 | 29317 | 19545 |


Runtime vs Number of Iterations as a Function of the Number of Processors (linear-linear)
4.E+05
3.E+05
3.E+05
2.E+05
2.E+05
1.E+05
5.E+04
0.E+00
Telapsed (seconds)
0.E+00
5.E+04
1.E+05
2.E+05
2.E+05
3.E+05
Number of Iterations
SerRef
Par4
Par8
Par16
Par32


Runtime vs Number of Iterations as a Function of the Number of Processors (log-log)
Telapsed (seconds)
1.E+06
1.E+05
1.E+04
1.E+03
1.E+02
1.E+01
1.E+00
1.E+00
1.E+01
1.E+02
1.E+03
1.E+04
1.E+05
1.E+06
Number of Iterations
SerRef
Par4
Par8
Par16
Par32


Runtime vs Number of Iterations as a Function of the Number of Processors (log-linear)
1.E+06
1.E+05
1.E+04
1.E+03
1.E+02
1.E+01
1.E+00
Telapsed (seconds)
0.E+00
5.E+04
1.E+05
2.E+05
2.E+05
3.E+05
Number of Iterations
SerRef
Par4
Par8
Par16
Par32


Speedup (Ts/Tp)
Speedup
Processor Elements
0
5
10
15
20
25
30
35
Speedup
Ideal


Efficiency = (Ts/Tp)/P = Ts/(P*Tp)
Efficiency (%)
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
5
10
15
20
25
30
35
Processors
Efficiency
Ideal

```
/* hello.c by James Otto, 1/31/11
-- for running serial processes
   on a cluster... see batch.hello */
#include <stdio.h>
#include <unistd.h>
int main(void)
{
  char cptr[100];
  gethostname(cptr,100);
  printf("hello, world from %s\n", cptr);
  return 0;
}
```

=================================

COMPILE & RUN SERIAL PGM

=================================

```
[tuckoo]$ mpicc -o hello hello.c
[mthomas@tuckoo ex.2014]$ mpirun -np 5 ./hello
hello, world from tuckoo
hello, world from tuckoo
hello, world from tuckoo
hello, world from tuckoo
hello, world from tuckoo
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "mpi.h"
int main (int argc, char* argv[])
{
  int rank, nprocs, ierr, i, error=0;
  MPI_Status status;

  ierr = MPI_Init(&argc, &argv);
  if (ierr != MPI_SUCCESS) {
    printf("MPI initialization error\n");}

  // processing element ID
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  // ID of communicator connecting PE's
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  printf("Hello Processor: rank:
       %d, nprocs: %d\n", rank, nprocs);

  MPI_Finalize();
  return 0;
}
```

=================================

COMPILE & RUN PARALLEL PGM

=================================

```
[tuckoo]$ mpicc -o hello_mpi hello_mpi.c
[tuckoo]$ mpirun -np 5 ./hello_mpi
Hello Processor: rank: 0, nprocs: 5
Hello Processor: rank: 1, nprocs: 5
Hello Processor: rank: 3, nprocs: 5
Hello Processor: rank: 4, nprocs: 5
Hello Processor: rank: 2, nprocs: 5
```

## Looptest demonstrates way to measure time app spends in subroutines

```
program looptest
! fortran 90 source code
implicit none
integer, parameter :: max=10000
integer            :: i, j
double precision   :: tws,twe, ts, te,
     a(max,max), x(max), y(max)
call cpu_time(tws)
!------ initialize arrays
a=0.0; x=0.0; y=0.0
do i=1,max
    x(i) = i;   y(i) = max-i
    do j=1,max
        a(i,j) = 10*j + i
enddo;   enddo
! -----compute  loop1
call cpu_time(ts)
call loop1(y,max)
call cpu_time(te)
print *,"Telap: loop 1 = ", (te - ts)
!------ compute  loop2
ts=0.0; te=0.0;
call cpu_time(ts)
call loop2(y,max)
call cpu_time(te)
print *,"Telap: loop 2 = ", (te - ts)
!------ compute  loop3
ts=0.0; te=0.0;
call cpu_time(ts)
call loop3(y,max)
call cpu_time(te)
print *,"Telap: loop 3 = ", (te - ts)
call cpu_time(twe)
print *,"Wallclock Time:  = ", (twe - tws)
```

```
contains
subroutine loop1(yloc,maxloc)
integer :: maxloc
double precision   :: yloc(maxloc)
do i=1,maxloc
    do j=1,maxloc
        yloc(i) = a(i,j) * x(j)
    enddo
enddo
end subroutine loop1

subroutine loop2(yloc,maxloc)
integer :: maxloc
double precision   :: yloc(maxloc)
do j=1,maxloc
    do i=1,maxloc
        yloc(i) = a(i,j) * x(j)
    enddo
enddo
end subroutine loop2

subroutine loop3(yloc,maxloc)
integer :: maxloc
double precision   :: yloc(maxloc)
do i=1,maxloc
    do j=1,maxloc
        yloc(i) = a(i,j) * sqrt(x(j))
    enddo
enddo
end subroutine loop3

end program looptest
```

## Compile with *gprof* option *(-p)*, and run job from command line

```
[mthomas@tuckoo]$ cat makefile
==============================
MAKE FILE
==============================
MPIF90 = mpif90
MPICC = mpicc
CC     = gcc
all: looptst looptstp
looptst: looptst.f90
        $(MPIF90) -o looptst looptst.f90

looptstp: looptst.f90
        $(MPIF90) -p -o looptstp looptst.f90

clean:
        rm -rf *.o looptst looptst-mpi
```

```
==============================
SERIAL JOB: FROM COMMAND LINE
==============================
[mthomas@tuckoo]$ ./looptstp
 Testing FORTRAN loops (column major):
 Telap: loop 1 =      960.8539      msec
 Telap: loop 2 =      580.9109      msec
 Telap: loop 3 =    1744.7349      msec
Wallclock Time: =     5861.1099      msec

==============================
PROFILING: using -p option in make
==============================
 [mthomas@tuckoo]$ gprof looptstp gmon.out
Flat profile:
Each sample counts as 0.01 seconds.
 cumulative    self              self      total
 time    seconds    seconds    calls    s/call    s/call   name
 37.58      1.39       1.39       1      1.39      3.67   MAIN__
 27.04      2.40       1.00       1      1.00      1.00   frame_dummy
 23.25      3.26       0.86       1      0.86      0.86   loop1.1529
 10.95      3.67       0.41       1      0.41      0.41   loop2.1523
```

## Run Serial Job In Queue

```
==========================
= SUBMIT SERIAL JOB TO QUEUE
==========================
[mthomas@tuckoo looptst]$ cat batch.looptstp
#!/bin/sh
#PBS -V
#PBS -l nodes=2:ppn=4:core4
#PBS -N looptstp
#PBS -joe
#PBS -q batch
cd $PBS_O_WORKDIR
NCORES=`wc -w < $PBS_NODEFILE`
echo "looptstp-test using $NCORES cores..."
mpirun -np 4 -hostfile $PBS_NODEFILE
                    --nooversubscribe ./looptstp
[mthomas@tuckoo looptst]$ !qsub
qsub batch.mpi-looptstp
16478.tuckoo.sdsu.edu
==========================
= OUTPUT (asynchronous)
==========================
 Telap: loop 1 =   0.84287199999
 Telap: loop 2 =   0.4549309999
 Telap: loop 2 =   0.455931
 Telap: loop 2 =   0.449931
 Telap: loop 2 =   0.455931
 Telap: loop 3 =   0.9918490
 Wallclock Time:        =      5.028235
 Telap: loop 3 =   0.99084
 Wallclock Time:        =      5.02623
```

```
------------------------------------------
 Telap: loop 1 =    0.8308729
 Telap: loop 1 =    0.8308739
 Telap: loop 1 =    0.8328739
 Telap: loop 1 =    0.8428719
------------------------------------------
 Telap: loop 2 =    0.4499310
 Telap: loop 2 =    0.4549309
 Telap: loop 2 =    0.4559310
 Telap: loop 2 =    0.4559310
------------------------------------------
 Telap: loop 3 =    0.9898489
 Telap: loop 3 =    0.9908489
 Telap: loop 3 =    0.9918490
 Telap: loop 3 =     1.0078469
------------------------------------------
 Wallclock Time:        =     5.02523599
 Wallclock Time:        =     5.0262349
 Wallclock Time:        =     5.02823599
 Wallclock Time:        =     5.049231
```

**Note: no gain by using multiple PE's $-- >$ no MPI calls in code**

## Add MPI Calls

```
program looptest
!
implicit none
include "mpif.h"
integer, parameter :: max=10000
double precision, allocatable
:: a(:,:), x(:), y(:)
double precision  :: tws,twe, ts, te
integer
:: i,j, rank, nprocs, ierr, token
integer           :: status(MPI_STATUS_SIZE)

call cpu_time(tws)
call MPI_INIT(ierr)
if (ierr .ne. MPI_SUCCESS) then
    print *, "Error: initing in MPI_INIT()"
    stop
endif

!— find out how many processes \&
local process rank
call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)

maxloc=gl_max/nprocs
allocate(a(maxloc,maxloc), x(maxloc),  &
            y(maxloc), stat=ierr)

!initialize arrays
do i=1,max
    x(i) = i; y(i) = max—i
    do j=1,max
        a(i,j) = 10*j + i
    enddo
enddo

! compute  loop1
call cpu_time(ts)
call loop1(y,max)
call cpu_time(te)
write(  )

! compute  loop2
ts=0.0; te=0.0;
call cpu_time(ts)
call loop2(y,max)
call cpu_time(te)
write(  )

! compute  loop3
ts=0.0; te=0.0;
call cpu_time(ts)
call loop3(y,max)
call cpu_time(te)
write(  )
call cpu_time(twe)
write(  )

call MPI_FINALIZE(ierr)

contains
 ..
```

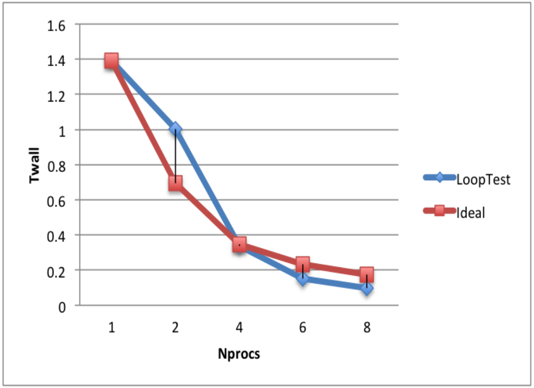
## Run MPI Job In Queue

```
==================
= SUBMIT JOB TO QUEUE
==================
[mthomas@tuckoo looptst]$ !qsub
qsub batch.mpi-looptstp
16478.tuckoo.sdsu.edu
==================
= OUTPUT (asynchronous)
==================
[mthomas@tuckoo looptst]$ cat mpi-looptstp.o16485
mpi-looptstp-test using 8 cores...
 LocaMAX:          2500
 LocaMAX:          2500
 LocaMAX:          2500
 LocaMAX:          2500
PE[ 0]: Telap, loop 1=      0.07698800
PE[ 3]: Telap, loop 1=      0.07698800
PE[ 2]: Telap, loop 1=      0.07598900
PE[ 2]: Telap, loop 2=      0.03799400
PE[ 1]: Telap, loop 3=      0.07998700
PE[ 1]: Telap, Twall=      0.33794800
PE[ 0]: Telap, loop 3=      0.07898800
PE[ 0]: Telap, Twall=      0.33594800
PE[ 3]: Telap, Twall=      0.34294600
PE[ 2]: Telap, loop 3=      0.07998800
PE[ 2]: Telap, Twall=      0.33294900
```

```
----------------
PE[ 0]: Telap, loop 1=      0.07698800
PE[ 1]: Telap, loop 1=      0.07698800
PE[ 2]: Telap, loop 1=      0.07598900
PE[ 3]: Telap, loop 1=      0.07698800
----------------
PE[ 0]: Telap, loop 2=      0.03799400
PE[ 1]: Telap, loop 2=      0.03799500
PE[ 2]: Telap, loop 2=      0.03799400
PE[ 3]: Telap, loop 2=      0.03699500
----------------
PE[ 0]: Telap, loop 3=      0.07898800
PE[ 1]: Telap, loop 3=      0.07998700
PE[ 2]: Telap, loop 3=      0.07998800
PE[ 3]: Telap, loop 3=      0.08098700
----------------
PE[ 0]: Telap, Twall=      0.33594800
PE[ 1]: Telap, Twall=      0.33794800
PE[ 2]: Telap, Twall=      0.33294900
PE[ 3]: Telap, Twall=      0.34294600
```

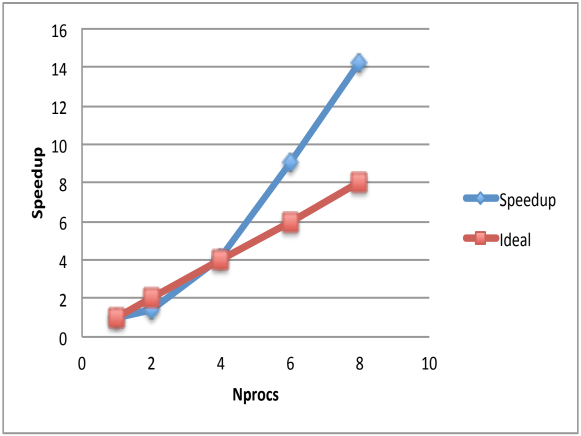
**Note: $T_{wall}$ reduced from 5+ seconds to 0.3**

## mpi-looptst RunTime (Twall)



Note: Ideal runtime computed using $T_{ideal} = \frac{T_{ser}}{p}$

## mpi-looptst: Speedup



## DEMO COLUMNS

LHS

RHS