CSCI 551
Numerical and Parallel Programming

Lecture 6-2 – MPI for Basic Integration

(Comparison of Thread and MPI Synchronization)

Reminders

- REVIEW on Tuesday next week, March 4th
  - Tues Activity code review for Mid-term
  - MPI Collective Comm focus and code in Chapter 3
  - Review of OpenMP and Pthreads code (as needed)

- Mid-term Thursday next week in discussion, March 6th
  - 70 minutes in class with 5 minute grace to wrap-up and turn in
  - 1 page of notes (both sides)
  - You can browse our site - csci-551
  - You can use your “sandbox” code you cloned from my github
  - You cannot use ChatGPT, Gemini, or any other AI assistance
  - You cannot consult any other person
  - If code you turn in is not your own, it is cheating if you don’t cite the source, and you may not get credit – write you own code!
  - No activity next week Thursday – enjoy!

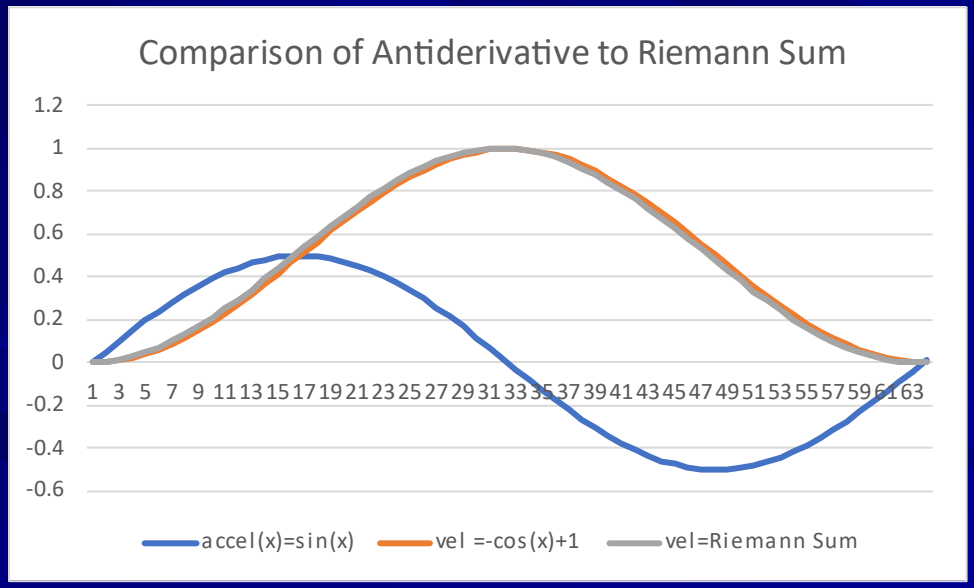
- Week-8 – Start Ex #4 (challenging) and discuss Final Parallel Program
  - Think over break (about Final Parallel program)
  - Enjoy, relax, recover, catch-up and tinker with Ex #4
  - Ex #4 due March 24th when we return (March 27th with 3-days grace)
  - Ex #5 can be last if you opt-in to present your Final Parallel Program

Major Goals for Week-6

- Wrap-up MPI Introductory Concepts (Week-6 & Week-7 Tues)
  - Review of History and Purpose (Alternatives)
  - Code Walk-throughs
  - Looking things Up about MPI
- Wrap-up material in part-1, introduction to Numerical and Parallel, and start review for Exam #1
  - Quiz #2 – Go over Quiz #2 solutions on Tuesday next week before Exam #1
  - Review notes and Q&A next week
- Start Ex #4 on Tuesday Week-8 – Not Due until After Break
- Review Tuesday Week-7, Exam #1 on Thursday Week-7 during class
  - One page of notes (both sides) – bring with you
  - Use Lab Computers to connect to ECC-Linux (or to run code locally)
  - You can surf web, but anything taken from web must be cited!
  - No outside help from another human
- Strategy after Exam #1
  - Focus on math and parallel applications – basic MPI, OpenMP, Pthreads, and numerical methods (meets ABET requirements)
  - Design for parallel programs – Divide and Conquer!
  - Introduce CUDA!
  - Emphasis on OpenMP and MPI, but Pthread options continued

Ex #3 Help – Not Linear but Well-Defined

- Ex #3 Train Problem is Non-Linear, but Simple sin(x) acceleration function (known anti-derivatives)
- Velocity has a definite integral solution of –cos(x)+C
- No need for Table Look-Up or interpolation
- Simple MPI implementation for n=1 … m processes
- Focus on Accuracy and Precision rather than speed-up



Antiderivative is Exact Solution (Spreadsheet)

Riemann or Trapezoidal is Numerical Approximation

Train acceleration profile is simple sine curve, but scaled for time and amplitude

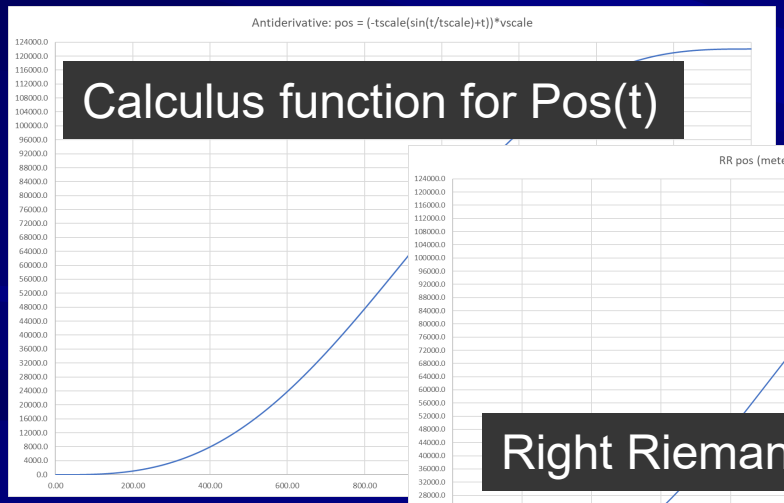
Non-Linear Functions

- Polynomials of degree 2 or higher
- Transcendental functions (trigonometric)
- Others (e.g., logarithmic), but these 2 are our primary interest
- Non-linear can be treated as “piece-wise linear”, but often can be directly implemented and solved with mathematics – e.g., antiderivative of sin(x)

- Linear systems include linear functions as well as linear systems of equations
- Non-linear functions and non-linear systems of equations are also of interest and we will study, but in less depth
- Ex #3 and Ex #6 are the main practice we have

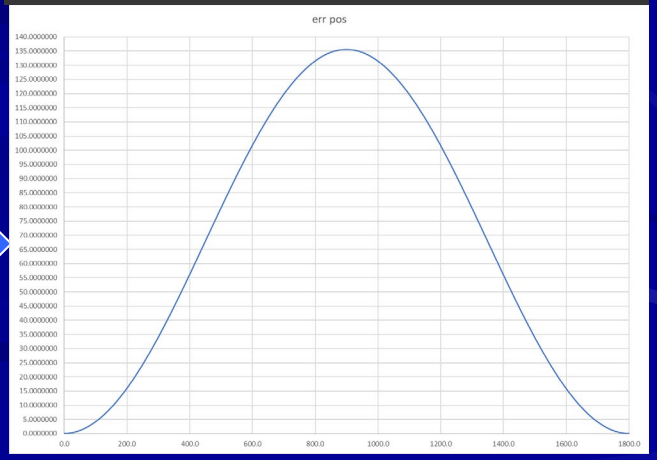
Most of the world is non-linear

- The world is mostly non-linear, engineers and scientists like to simplify with linear models (do you agree?)
- Potential accuracy issues – e.g., linear interpolation with non-linear functions
- In Ex #3 we use direct function generator rather than linear interpolation (Ex #4)
- Ex-3-Smooth-Function-Train-profile.xlsx



Diff?

About 135 meters in error



Cluster with specific nodes - Scaling

- running-an-mpi-program.html

```
c1_hosts  c2_hosts  greetings  greetings.c  Makefile  README
(base) sbsiewert@o251-14:~/code/hello_custer$ mpirun -n 16 -ppn 2 -f c1_hosts ./greetings
Hello from process 0 of 16 on o251-14
Hello from process 1 of 16 on o251-14
Hello from process 2 of 16 on o251-15
Hello from process 3 of 16 on o251-15
Hello from process 4 of 16 on o251-16
Hello from process 5 of 16 on o251-16
Hello from process 6 of 16 on o251-17
Hello from process 7 of 16 on o251-17
Hello from process 8 of 16 on o251-18
Hello from process 9 of 16 on o251-18
Hello from process 10 of 16 on o251-19
Hello from process 11 of 16 on o251-19
Hello from process 12 of 16 on o251-20
Hello from process 13 of 16 on o251-20
Hello from process 14 of 16 on o251-21
Hello from process 15 of 16 on o251-21
(base) sbsiewert@o251-14:~/code/hello_custer$
```

2 on 14

2 on 15

2 on 16

2 on 17

2 on 18

2 on 19

2 on 20

2 on 21

```
# my birthyear is odd, so I'm on cluster 251,
# using my birthdate node and 7 adjacent nodes
o251-14
o251-15
o251-16
o251-17
o251-18
o251-19
o251-20
o251-21
o251-22
o251-23
o251-24
o251-25
```

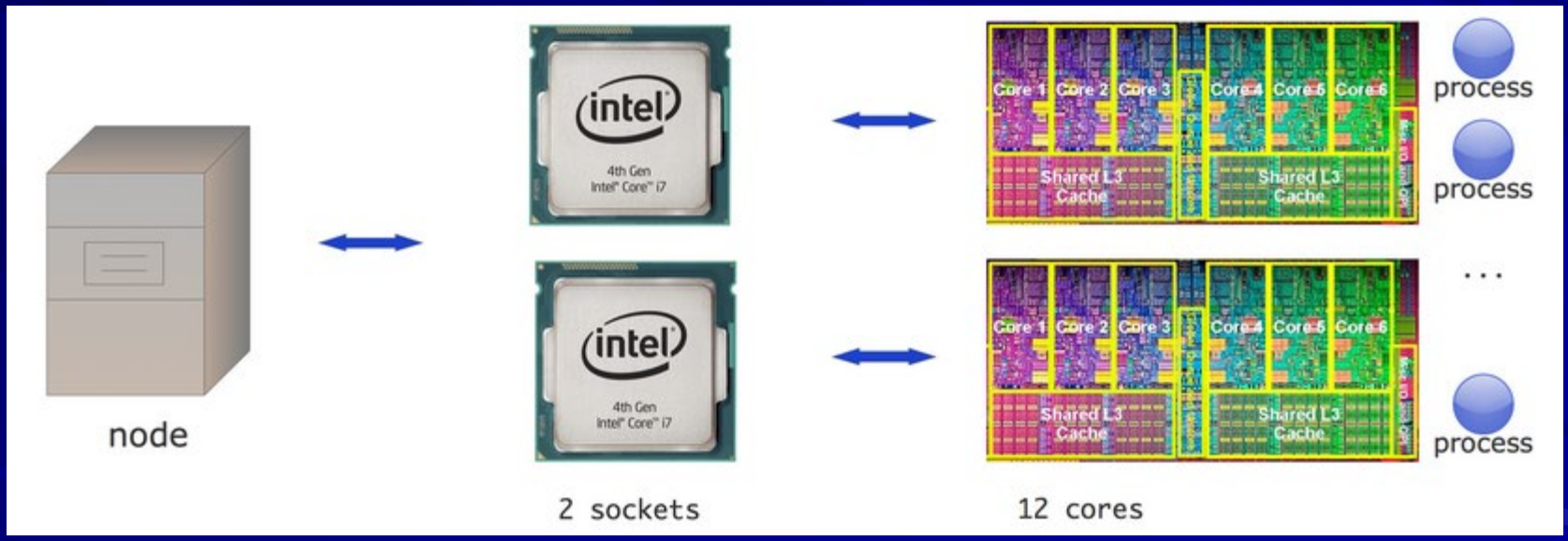
use up to 8 hosts nodes specified

Documentation

1) Intel PS XE – mpi-library-documentation-overview.html
2) OpenMPI - https://www.open-mpi.org/doc/current/

Recall Scale-up – Multi-programming

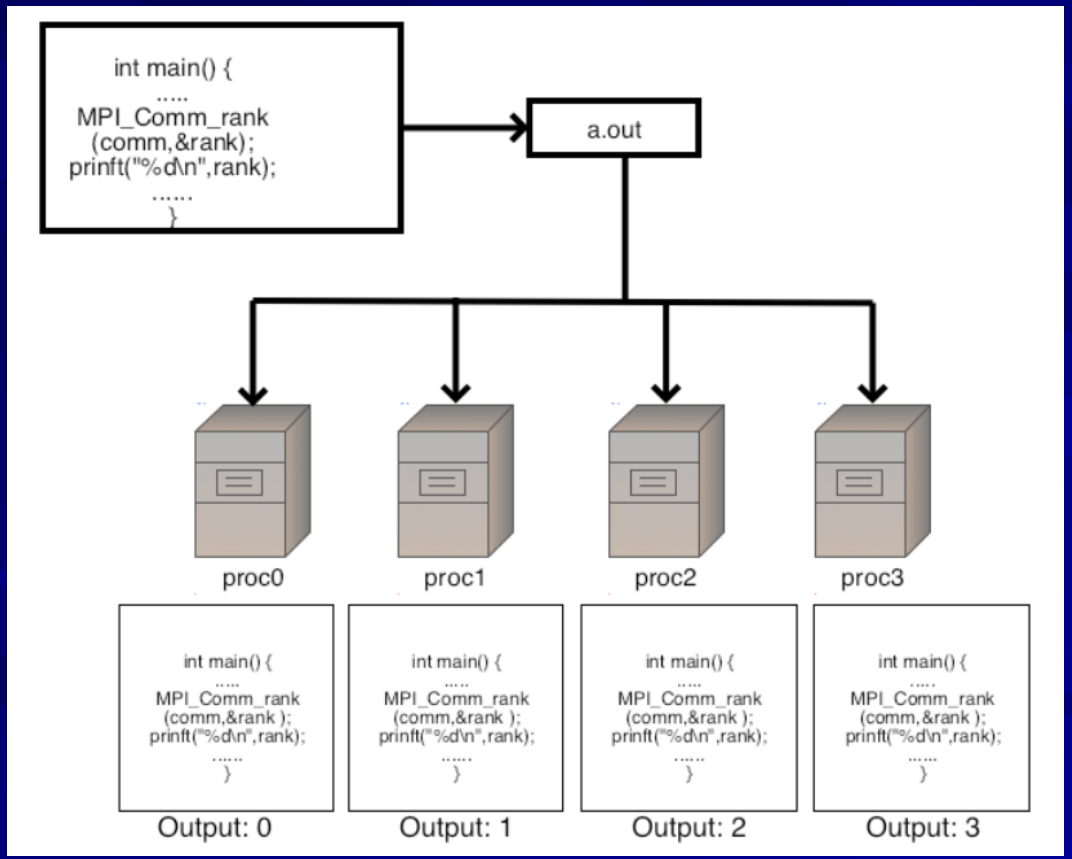
- Scaling up a single node and load balancing processes
  - Separate address space for each process (use messages to communicate and synchronize)
  - Processes run on multiple CPUs and processor cores



https://pages.tacc.utexas.edu/~eijkhout/pcse/html/mpi-functional.html

Scale-out to Multiple Nodes with MPI

- As I create more processes to divide up work, assign each process to Node, CPU, and core
- Use MPI to divide up work, identify role, merge results



Linux nodes

- Each node schedules processes and provides load balancing
- SMP UMA/NUMA nodes
- Details of node selection and CPU/core selection are provided by system
- Cluster + OS
- 1800 doubles broadcast with MPI_Bcast in 137 µsec on gigE, 13.7 µsec on 10GE, 1.37 µsec 100GE

gigE CSU - Gigabit_Ethernet
10GE - 10_Gigabit_Ethernet
100GE - 100_Gigabit_Ethernet

Scale-Out – The Extremes

- Top-500 - https://top500.org/
  - November 2024 – El Capitan, HPE Cray with 11,039,616 cores
  - **"El Capitan** has achieved the top spot and is officially the third system to reach exascale computing after Frontier and Aurora"
- NSF ACCESS – https://access-ci.org/
  - Get an ID if you want to work on Quantum vs Parallel Advantage
  - Graduate students can apply to be a PI
- Infiniband Interconnection - 800 Gbps, full duplex
  - Our cluster – 1 Gbps, full duplex
  - https://www.nvidia.com/en-us/networking/infiniband-switching/
  - 800 Gbps switch
- MPI has in theory infinite scale-out
  - Limited by speed of light and power to run the system
  - Cost to build a networked cluster of nodes and the interconnection

A Brief History of Scale-Out

- Distributed Computing – networking PCs, Workstations, Mainframes, mini-computers (1980's)

- Embarrassingly Parallel was at first distributed
  - E.g., 10,000 Monte Carlo cases run on 100 machines, 100 at a time, overnight, using "rsh", trusted hosts, and scripts with programs

- RPC – Remote Procedure Call (1980's)
- PVM – Parallel Virtual Machine (1990's)
- Clusters and MPI (1994 v. 1.0)
- OpenMP & Pthreads Shared Memory (mid to late 1990's)

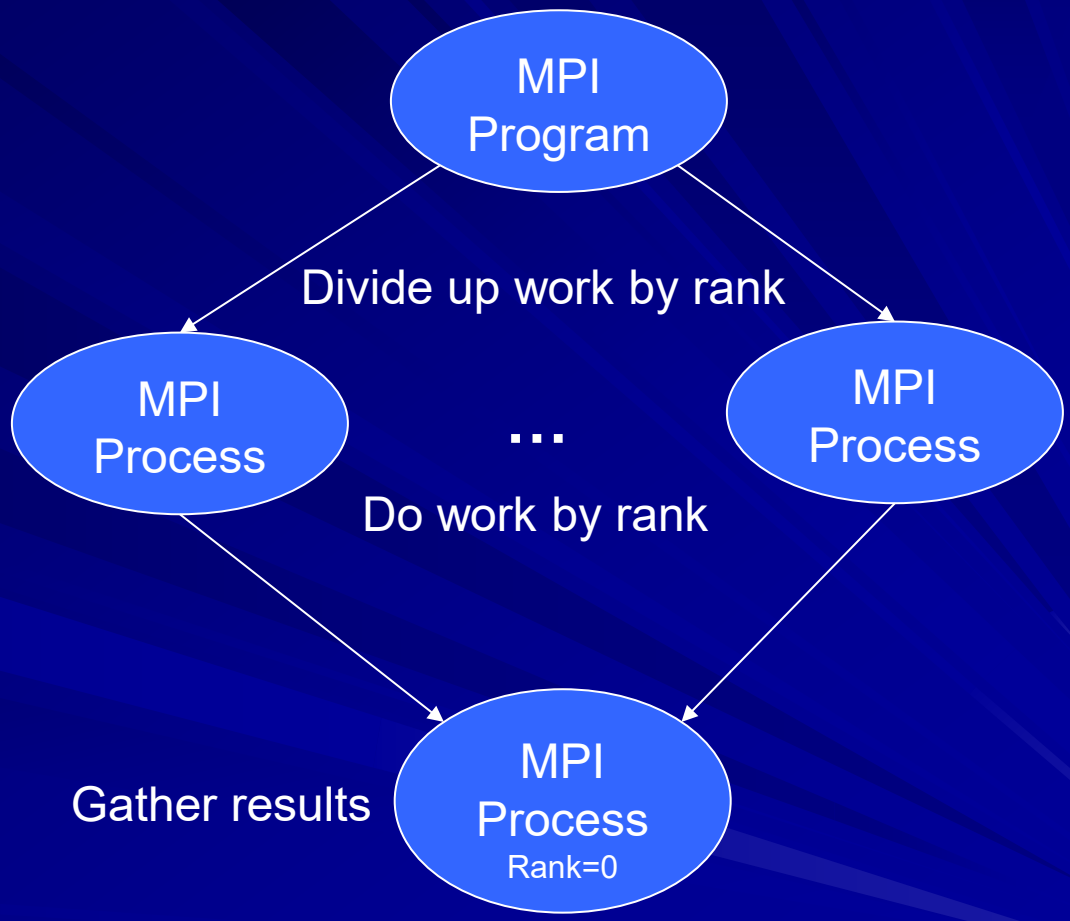
Shuttle - used Right Riemann Sum in flight software (simple), but verified with Runge-Kutta integration of systems of differential equations with a ground distributed system (rsh scripts)

Sun Microsystems – networks of workstations (1985-1995) – most often, single core

MPI Concepts for Parallel Work

- Create “n” clones of the same program (process)
- Start them on the same or different networked machines
- Divide up work by rank
- Rank 0 gathers results from other ranks, reports result (e.g., MPI_Reduce)

- Complex Example (Ex #4)
  - Loop carried dependency
  - Must use dynamic programming strategy
  - See compare.c



Simple Divide and Conquer!

MPI Major Concepts Reviewed

- Used for Distributed Memory Parallel Processing
- Abstracts the details of process creation and message passing
- Abstracts the details of distributing work and gathering results
- Provides efficient methods to do so
- Helps programmer to maximize P section and minimize (1-P) for any scale of S in Amdahl's law

Methods of Synchronization – Parallel

- For MPI – main method of synchronization is MPI_Send, MPI_Recv, and Collection Communications
- Barriers, condition variables, spin locks, signals, & message queues

Platform	Operating Environment	Semaphores	Locks	Other	Composite
Pthreads on: Multi-core OS	Linux SMP & POSIX	sem_t, sem_init (sem_post, sem_wait)	pthread_mutex_init pthread_mutex_lock, pthread_mutex_unlock	<stdatomic.h> e.g., atomic_int pthread_barrier_init pthread_spin_lock mq_open mq_send mq_receive POSIX signals: sigqueue, sigaction	pthread_cond_init pthread_create pthread_join
MPI on: Fabric or Torus interconnected Linux node cluster	Linux cluster & OpenMPI MPI_Comm_size MPI_Group_size	N/A – messages can be blocking or non-blocking	N/A – no shared memory	MPI_Barrier MPI_Send MPI_Recv	MPI_Wait MPI_Waitany MPI_Bcast MPI_Scatter MPI_Gather

1) Gallmeister, Bill. *POSIX. 4 Programmers Guide: Programming for the real world.* " O'Reilly Media, Inc.", 1995.
2) Stevens, W. Richard, and Stephen A. Rago. *Advanced programming in the UNIX environment*. Addison-Wesley, 2008.
3) POSIX Standards: https://publications.opengroup.org/, http://get.posixcertified.ieee.org/
4) http://www.rc.usf.edu/tutorials/classes/tutorial/mpi/

Locks and Semaphores – ideally avoid using them whenever possible for parallel programs.

For Shared Data - Keep it Simple!

- Data and computational dependencies – e.g., simulation and integration of state to get next state
  - Notes-on-Two-Pass-Parallel-Integration-by-Sub-interval.pdf
  - See OpenMP Speed-Up Example - timeprofiles_omp.c

- Shared memory with multiple readers and writers (**can be complex**)
  - MUTEX (e.g., example-mutex-demos/)
  - Producer/Consumer (e.g., Writer-Reader-sync-demo/)
  - **Thread indexed data and use of stack vs. global memory – keep it simple!**

- High degree (polynomial bounded, but complex) – NC, P-complete
- High dimensionality – 3D and beyond
  - 2D Video – X,Y, color, time (4 dimensions)
  - 3D Graphics
  - Simulation of physical systems (e.g., 6 DOF robotics)
  - Data analytics
- Non-Linear and Linear Systems (many equations and unknowns)
- High Fidelity, Resolution and Long Duration (faster than real-time)
- Large search spaces and NP problems

MPI_Comm_size, MPI_Comm_rank

- mpi_hello.c
- Point-to-Point communication with messages (synchronized)
- Major Concepts
  - Order of messages
  - What happens with MPI_Recv and no corresponding MPI_Send?
  - Message Matching
    - Recv_comm = Send_comm
    - Recv_tag = Send_tag
    - Dest = process_r, Src = process_q

```
MPI_Send(greeting,
         strlen(greeting)+1,
         MPI_CHAR, 0, 0,
         MPI_COMM_WORLD);

MPI_Recv(greeting,
         MAX_STRING,
         MPI_CHAR, q, 0,
         MPI_COMM_WORLD,
         MPI_STATUS_IGNORE);

See p. 91 in Pacheco
```

```
MPI_Comm_rank

My ID within a Comm world

0 is special

Normally 0 deals with input
Normally 0 deals with
distributing work and
gathering results

Other ranks output only
```

More MPI Code Examples

- mpi_many_msgs.c – use of MPI_Barrier, MPI_Send, MPI_Recv, and time-stamping for performance
  - A simple benchmark
  - What does it tell us?
  - What are the limitations of this example?
- mpi_mat_vect_time.c – adds use of MPI_Reduce
  - A simple benchmark
  - What does it tell us?
  - What are the limitations of this example?
- mpi_mat_vect_mult.c – adds use of MPI_Scatter, MPI_Gather, MPI_Allgather, MPI_Allreduce

MPI Code:

- Discussion
  - API Docs
  - API "man"
  - Program structure
- Benchmarks
- Examples
  - Pacheco
  - Siewert
  - Quinn

Comparison Discussion

- Which provides most cost-effective and efficient scaling?
  - Distributed memory, cluster scaling with MPI
  - Shared memory, CPU and core scaling with Pthreads or OpenMP
  - Hybrid using both?
- Which is simpler to program?
- Which is easer to use for speed-up?
- Which can be scaled the most?
  - In terms of number of processors and processes/threads

MPI Continued

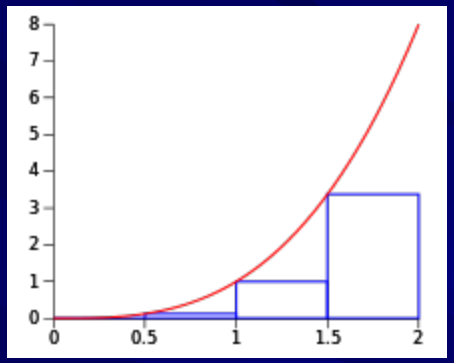
- Complete reading of Pacheco Chapter 3
  - Examples for serial and parallel portions of code
  - Parallel integration example
  - Parallel sort example
- Focus on Assignment #3
- I will go over simple demonstration and background on MPI to make integration parallel (not solution)
  - Design Concept - Two-Pass-Parallel-Integration-by-Sub-interval.pdf
  - Key is function modeling and data sharing
  - Sharing a read-only array is simpler with OpenMP or Pthreads
  - Sharing a read-only array LUT is simple for acceleration
  - For velocity array, must share a new table (velocity LUT) or at least initial conditions

Completing Exercise #3

- Read Pacheco chapter 3 through p. 113
  - See Intel documents on MPI
- Simple MPI program – based on book to explore MPI and cluster features
- Top Down – MPI examples (csci551/code/hello_cluster/ , MPI_Examples/)
  - Build, run, observe
  - Review code and explain (see book for MPI_Examples
- Bottom Up – Methods of integration
  - Simple forward integration, large, medium, small step size
  - Trapezoidal integration, large, medium, small step size
- Application – Train Acceleration and Braking
  - Create a profile of position and velocity over time
  - Given acceleration timeline
  - Compute arrival time
  - Use MPI to speed up the analysis (6 runs, “n” segments to integrate)

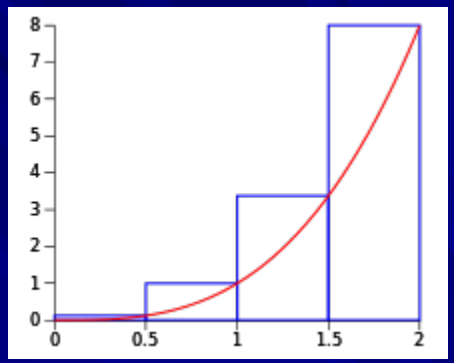
Methods of Integration (Riemann Sum)

Left Riemann Sum (Forward)
A_n= f(a) dx
dx=[b-a]/n



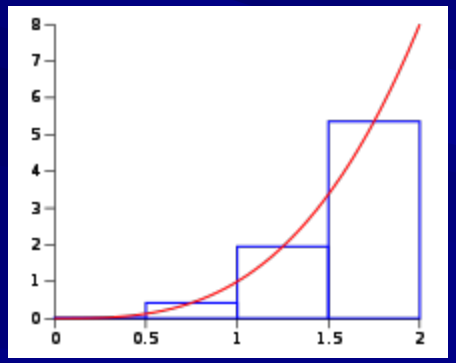
A_{b-a} = dx [(f(a) + f(a+dx) + f(a+2dx) + … + f(b-dx)]

Right Riemann Sum (Backward)
A_n= f(a+dx) dx
dx=[b-a]/n



A_{b-a} = dx [f(a+dx) + f(a+2dx) + … + f(b)]

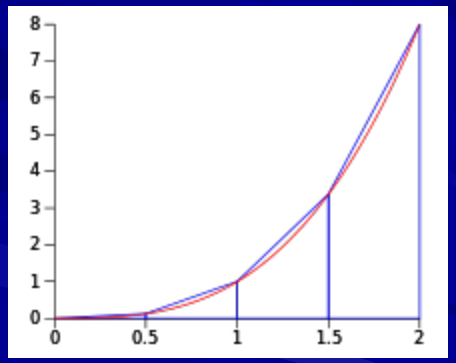
Midpoint Rule
A_n= f(a+dx/2) dx
dx=[b-a]/n



A_{b-a} = dx [f(a+dx/2) + f(a+3dx/2) + … + f(b-dx/2)]

Trapezoidal Rule

A_{trap}=½ h (f(a)+f(b))
dx=[b-a]/n
A_n=½ dx (f(a)+ f(a+dx))

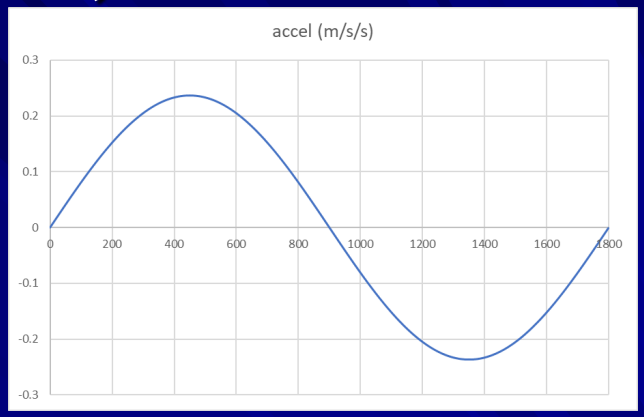


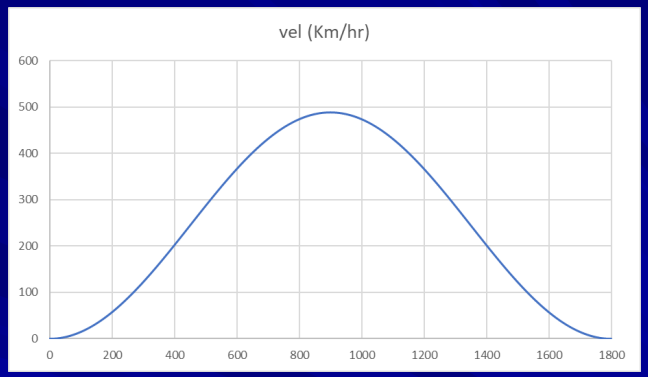
½ dx [(f(a) + f(a+dx)) + (f(a+dx) + f(a+2dx)) …]
½ dx [f(a) + 2f(a+dx) + 2f(a+2dx) + … f(b)]

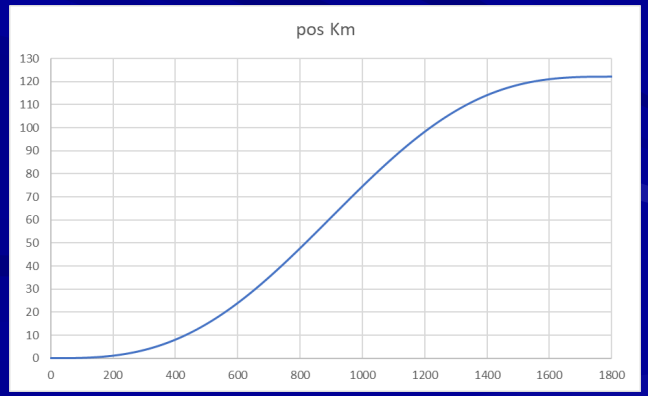
Note: impact of function – convex or concave on accuracy of result

Train Profile – Non-Linear, but Smooth

- Acceleration(t) is non-linear sine curve
- V(t) is integral of A(t) 0…1800
- P(t) is integral of V(t) 0…1800
- Sine is math function (no need for interpolation of modeling of piecewise linear functions)

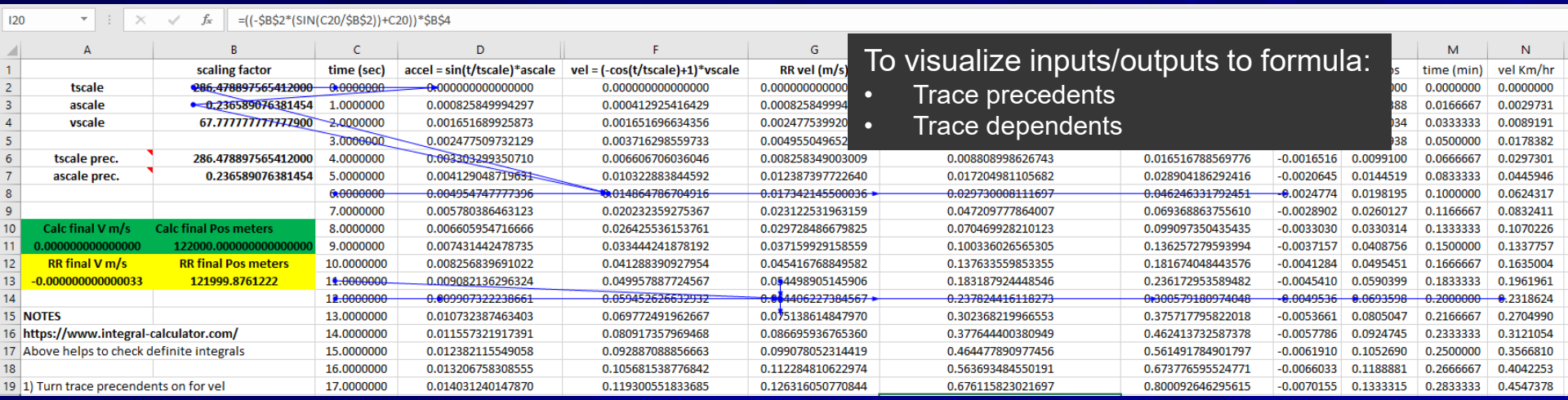






Train Problem

- As given in spreadsheet, this is 1 second step size, Right-Riemann Sum (could modify to make it Left Riemann)



	A	B	C	D	F	G	M	N
1		scaling factor	time (sec)	accel = sin(t/tscale)*ascale	vel = (-cos(t/tscale)+1)*vscale	RR vel (m/s)	time (min)	vel Km/hr
2	tscale	286.478897565412000	0.0000000	0.000000000000000	0.000000000000000	0.000000000000	0.0000000	0.0000000
3	ascale	0.236589076381454	1.0000000	0.000825849994297	0.000412925416429	0.000825849994	0.0166667	0.0029731
4	vscale	67.777777777777900	2.0000000	0.001651689925873	0.001651696634356	0.002477539920	0.0333333	0.0089191
5			3.0000000	0.002477509732129	0.003716298559733	0.004955049652	0.0500000	0.0178382
6	tscale prec.	286.478897565412000	4.0000000	0.003303299350710	0.006606706036046	0.008258349003009	0.0666667	0.0297301
7	ascale prec.	0.236589076381454	5.0000000	0.004129048719631	0.010322883844592	0.012387397722640	0.0833333	0.0445946
8			6.0000000	0.004954747777396	0.014864786704916	0.017342145500036	0.1000000	0.0624317
9			7.0000000	0.005780386463123	0.020232359275367	0.023122531963159	0.1166667	0.0832411
10	Calc final V m/s	Calc final Pos meters	8.0000000	0.006605954716666	0.026425536153761	0.029728486679825	0.1333333	0.1070226
11	0.000000000000000	122000.000000000000000	9.0000000	0.007431442478735	0.033444241878192	0.037159929158559	0.1500000	0.1337757
12	RR final V m/s	RR final Pos meters	10.0000000	0.008256839691022	0.041288390927954	0.045416768849582	0.1666667	0.1635004
13	-0.000000000000033	121999.8761222	11.0000000	0.009082136296324	0.049957887724567	0.054498905145906	0.1833333	0.1961961
14			12.0000000	0.009907322238661	0.059452626632932	0.064406227384567	0.2000000	0.2318624
15	NOTES		13.0000000	0.010732387463403	0.069772491962667	0.075138614847970	0.2166667	0.2704990
16	https://www.integral-calculator.com/		14.0000000	0.011557321917391	0.080917357969468	0.086695936765360	0.2333333	0.3121054
17	Above helps to check definite integrals		15.0000000	0.012382115549058	0.092887088856663	0.099078052314419	0.2500000	0.3566810
18			16.0000000	0.013206758308555	0.105681538776842	0.112284810622974	0.2666667	0.4042253
19	1) Turn trace precendents on for vel		17.0000000	0.014031240147870	0.119300551833685	0.126316050770844	0.2833333	0.4547378

	F	G	M	N
1				
2	0.000000000000000	0.000000000000	0.0000000	0.0000000
3	0.000412925416429	0.000825849994	0.0166667	0.0029731
4	0.001651696634356	0.002477539920	0.0333333	0.0089191
5	0.003716298559733	0.004955049652	0.0500000	0.0178382
6	0.006606706036046	0.008258349003009	0.0666667	0.0297301
7	0.010322883844592	0.012387397722640	0.0833333	0.0445946
8	0.014864786704916	0.017342145500036	0.1000000	0.0624317
9	0.020232359275367	0.023122531963159	0.1166667	0.0832411
10	0.026425536153761	0.029728486679825	0.1333333	0.1070226
11	0.033444241878192	0.037159929158559	0.1500000	0.1337757
12	0.041288390927954	0.045416768849582	0.1666667	0.1635004
13	0.049957887724567	0.054498905145906	0.1833333	0.1961961
14	0.059452626632932	0.064406227384567	0.2000000	0.2318624
15	0.069772491962667	0.075138614847970	0.2166667	0.2704990
16	0.080917357969468	0.086695936765360	0.2333333	0.3121054
17	0.092887088856663	0.099078052314419	0.2500000	0.3566810
18	0.105681538776842	0.112284810622974	0.2666667	0.4042253
19	0.119300551833685	0.126316050770844	0.2833333	0.4547378

- You can do just as well than spreadsheet (better for complex functions), but smoothness and symmetry minimizes error and improves accuracy
- Your accuracy can be better based on smaller step size and/or better method (e.g., Trapezoidal)

Function Generator for Acceleration

- Option #1 – Most Generally use a Look-Up Table with Linear interpolation
  - Advantages: Works for any function given, assumes piecewise linear approximation of real function, and provides accel(time) for any time you need. Works for integration of derived velocity(time) as well
  - Disadvantages: Piecewise linear interpolation is an approximation for non-linear functions and works best for smooth continuous functions.
- Option #2 – If Acceleration(time) is a known mathematical function, model it directly using "-lm" Math Library functions (e.g., sine)
  - Advantages: As accurate as the math library implementation, simple, direct model
  - Disadvantages: Works for accel(time), but what about velocity(time)? Only works well for second integration if first has a known anti-derivative