An Introduction to Parallel Computing:
Portable Shared Memory Parallel Programming with OpenMP

Marty Kandes, Ph.D.

HPC User Services Group
San Diego Supercomputer Center
University of California, San Diego

Student Cluster Competition Training Session
Friday, April 26th, 2019
1:00PM - 3:00PM PT

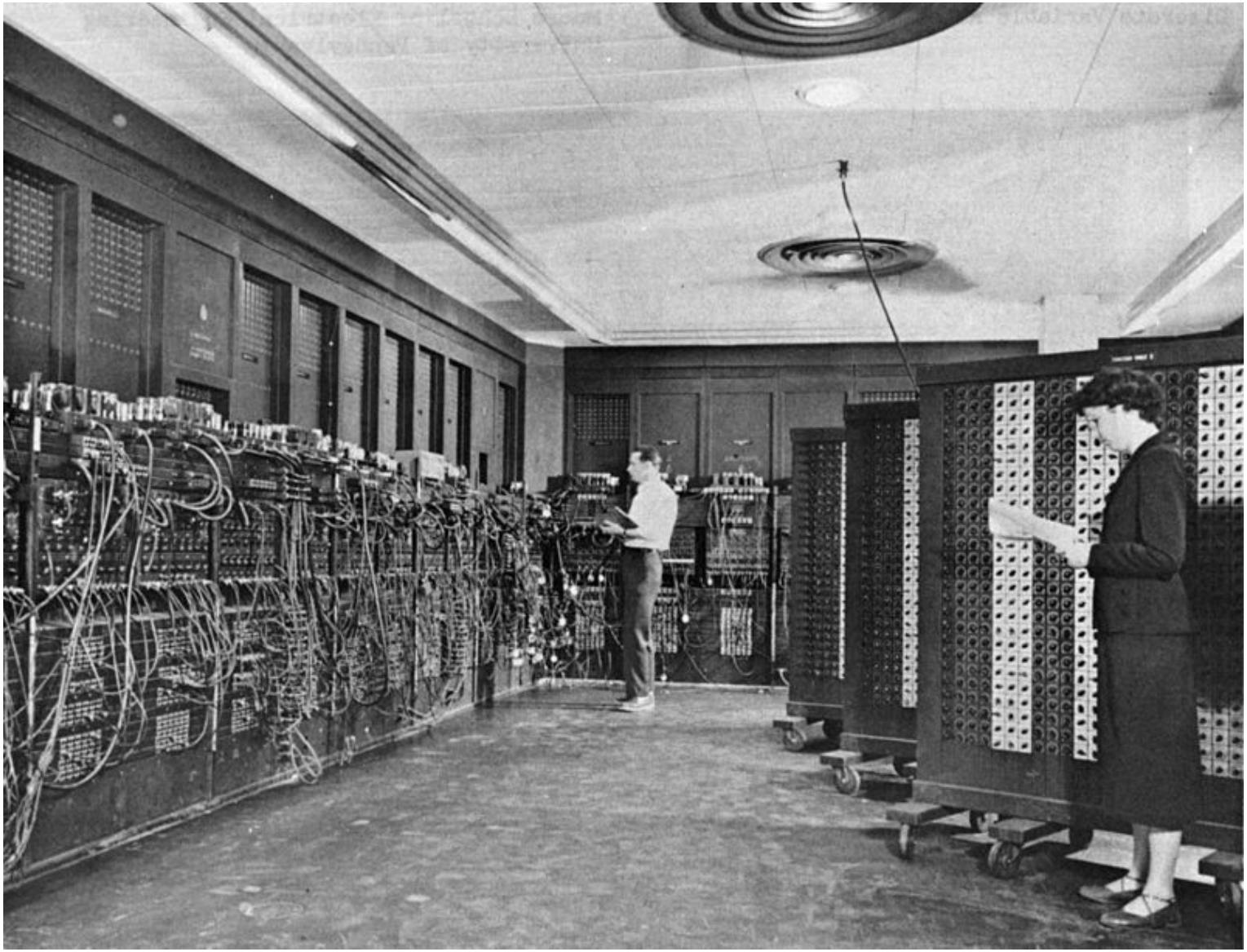
About Me

- B.S. in Physics + Applied Mathematics (Michigan); undergraduate research: experimental physics
- M.S. in Physics, Ph.D. Computational Science (SDSU); graduate research: computational/theoretical physics
- Started working in Data Center Operations Group @ SDSC while finishing Ph.D.
- Previously worked for Distributed High-Throughput Computing Group @ SDSC and the Open Science Grid
- Joined HPC User Services Group @ SDSC in April 2017

An Overview of Today's Training Session

- A Brief History of Computation
- Serial Computation and its Limits
- Parallel Computation and its Limits
- An Introduction to OpenMP
- Advanced Code Optimization with OpenMP
- Q&A?

A Brief History of Computation

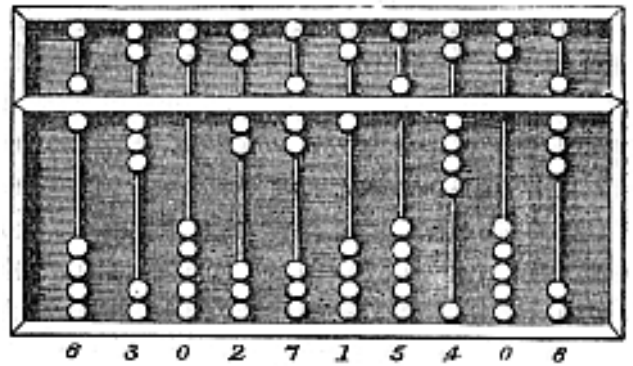


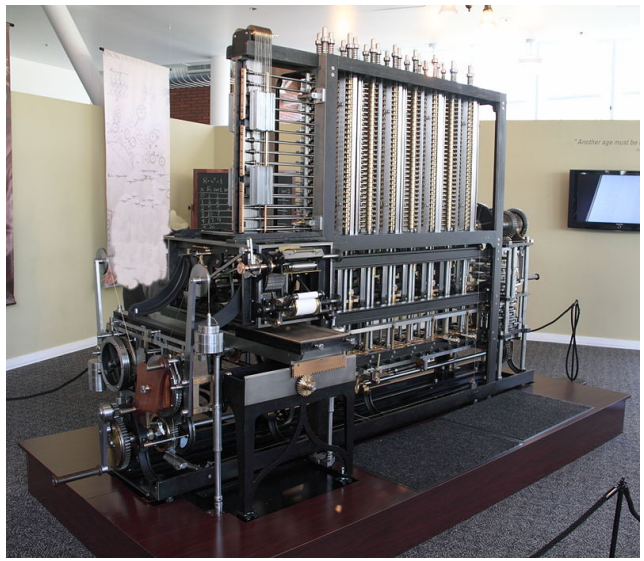
What is a Computer?

- A **computer** is a machine that can be instructed to carry out a sequence of operations – often arithmetic or logic operations — to accomplish a specific task or set of tasks.
- A computer **program** is the set of instructions or operations that perform a specific task or set of tasks executed by a computer.

The Mechanical Age of Computing

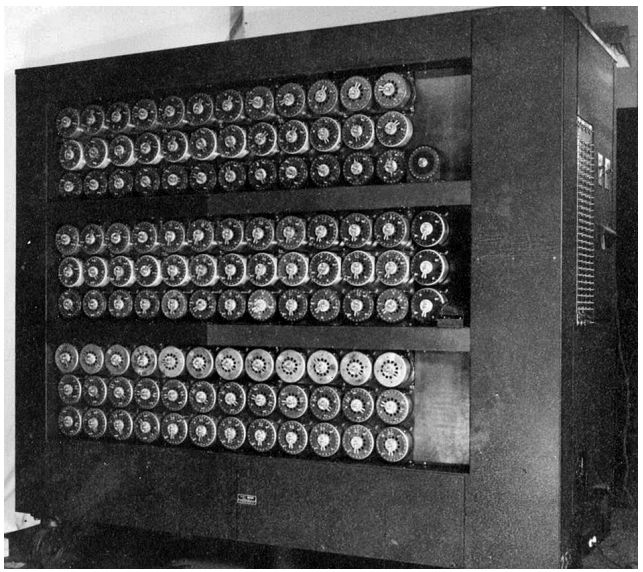
- 2700-2300 BCE: Earliest known use of the abacus
- 1050-771 BCE: Earliest known use of an analog computer
- 1642: First mechanical calculator
- 1725: First punch card system
- 1837: First design for a general-purpose computer
- 1843: First design of a computer program





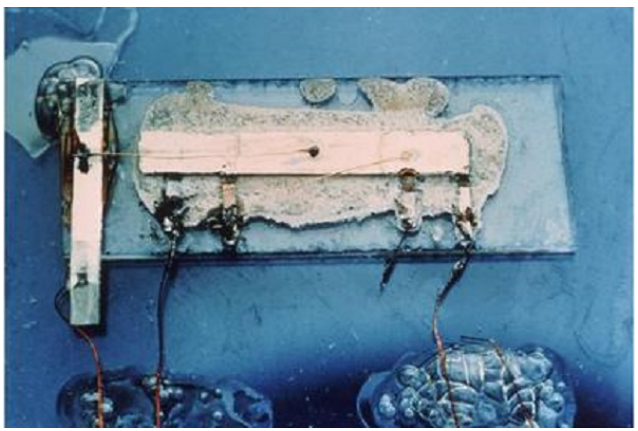
The Electro-Mechanical Age of Computing

- 1890: First use of a punch card system to compute the U.S. Census
- 1911: Founding of International Business Machines Corporation
- 1939: First Polish bombe begins deciphering German Enigma messages



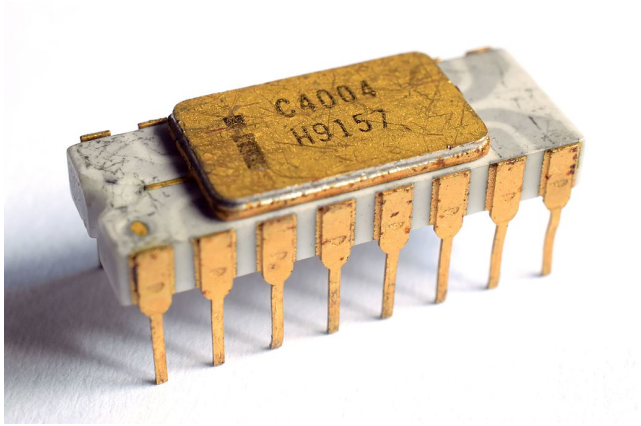
The Electronic Age of Computing

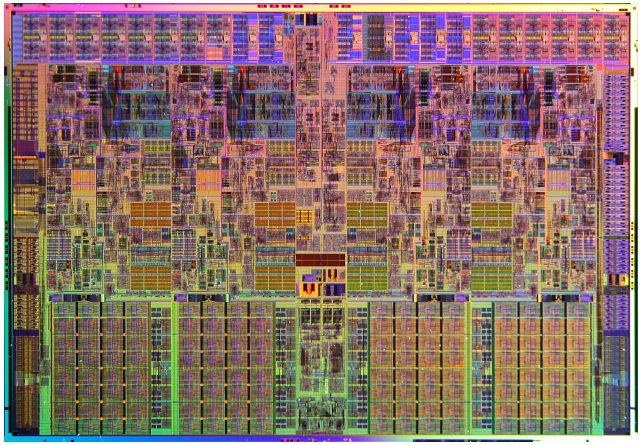
- 1936: First description of a universal Turing machine
- 1945: First electronic, general-purpose computer; von Neumann computer architecture
- 1947: First transistor
- 1951: First parallel computer
- 1953: First high-level computer programming language
- 1958: First integrated circuit; First computer network



The Electronic Age of Computing

- 1961: First all-transistor supercomputer
- 1965: First description of Moore's Law
- 1969: First link of ARPANET established
- 1971: First microprocessor
- 1973: First public release of the Unix operating system
- 1989: First public Internet service provider





Serial Computation and its Limits

Parallel Computation and its Limits

...

https://computing.llnl.gov/tutorials/parallel_comp/

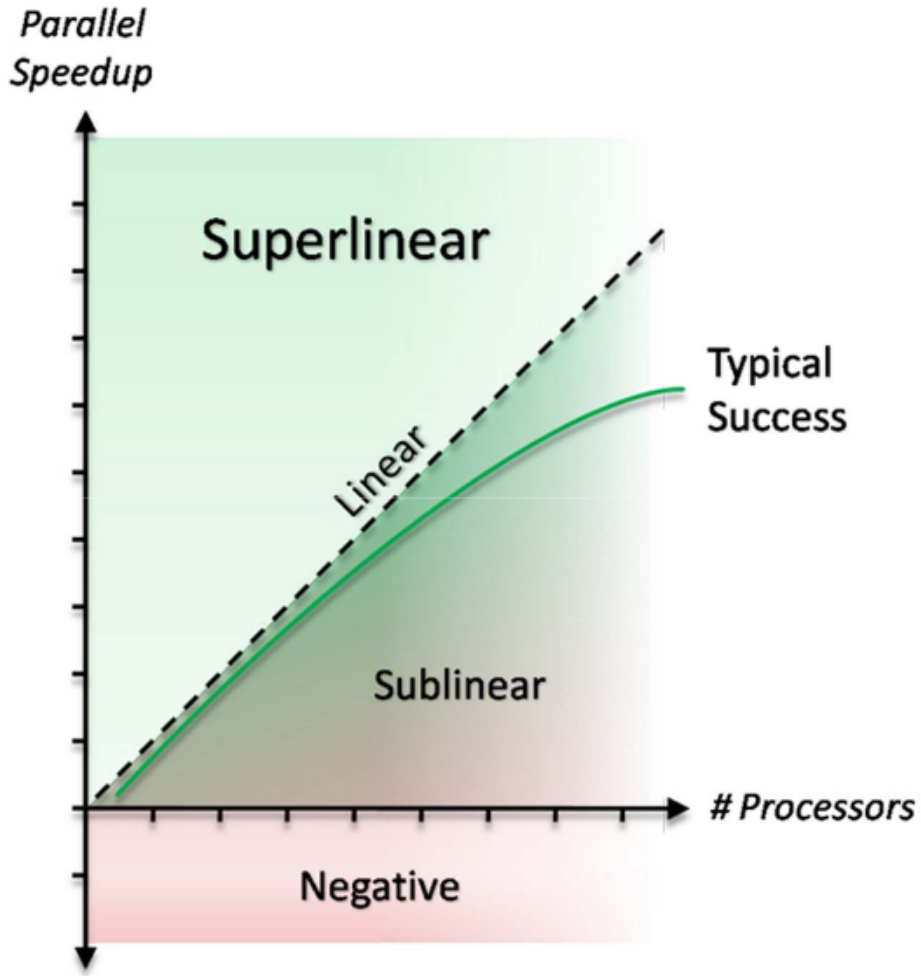
Parallel Speedup

- Parallel speedup measures how much faster a parallel algorithm solves a given problem compared to its sequential version.
- Parallel speedup is defined as

$$S_n = \frac{T_1}{T_n},$$

where T_1 is the execution runtime of the sequential algorithm on a single processing unit and T_n is the execution runtime of the parallel algorithm on n processing units.

- In general, we expect $T_1 > T_n$ such that $S_n > 1$.



Parallel Efficiency

- Parallel efficiency measures how well-utilized the hardware processing units are in solving a given problem.
- Parallel efficiency is defined as

$$E_n = \frac{S_n}{n} = \frac{T_1}{nT_n}.$$

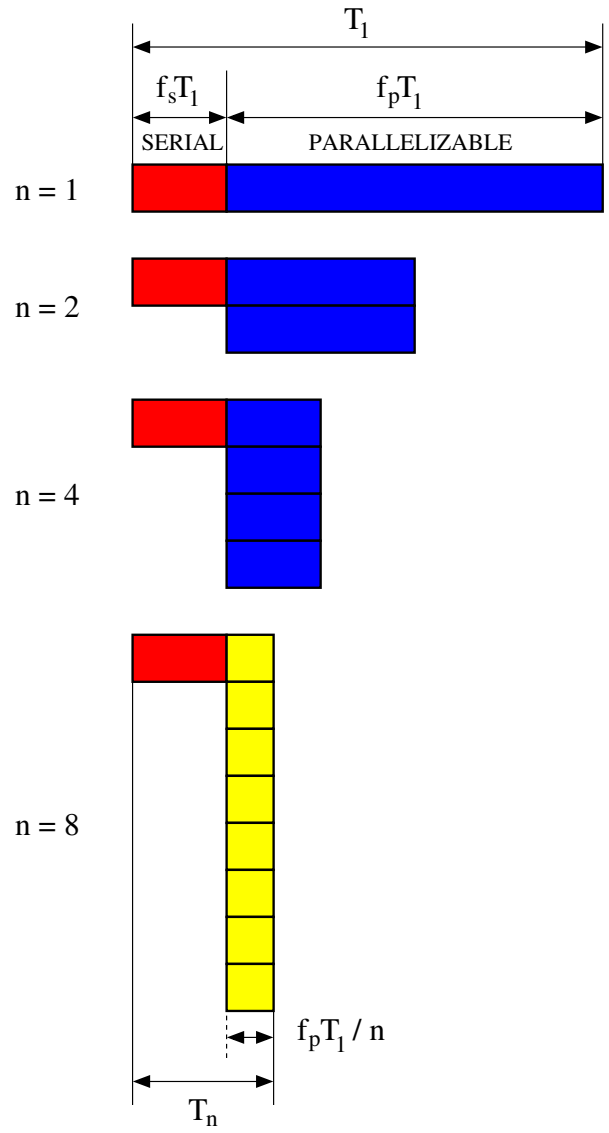
- Therefore, $0 \leq E_n \leq 1$ measures the fraction of time a processing unit performs useful work.

Strong Scalability and Amdahl's Law

- Strong scalability measures how the time-to-solution for a problem varies as a function of the number of processing units when the *problem size is fixed*.
- Strong scalability is limited by Amdahl's law, which states that the theoretical speedup of a parallel algorithm is given by

$$S_n = \frac{1}{f_s + \frac{f_p}{n}},$$

where f_p is the fraction of an algorithm's instructions that can be executed in parallel, while f_s is the fraction of instructions that must be executed sequentially.



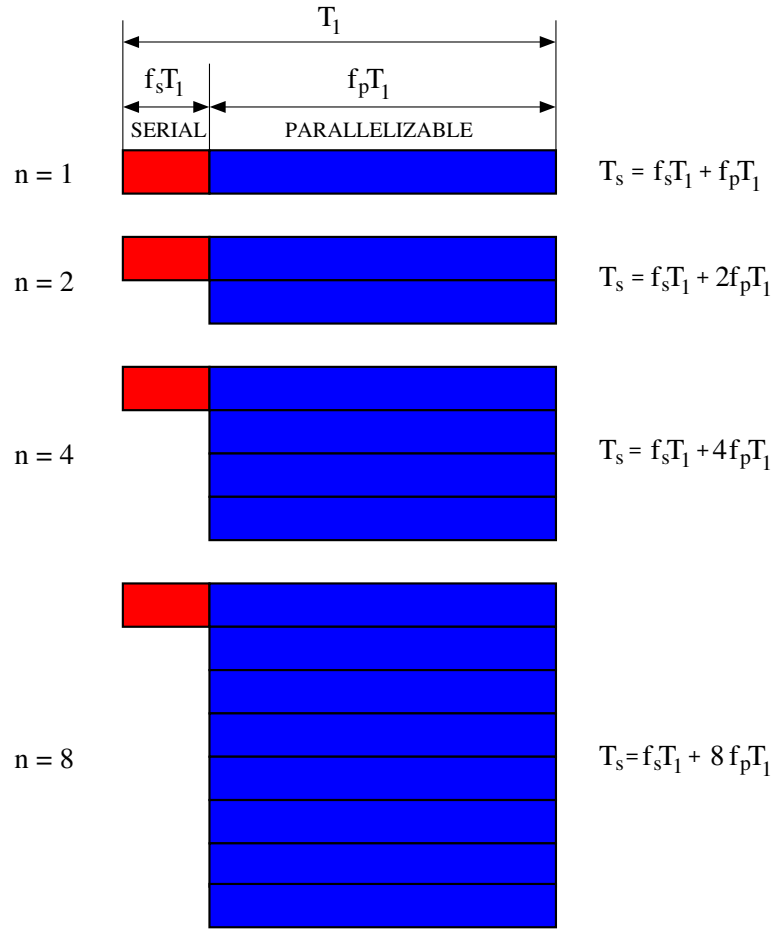
Weak Scalability and Gustafson's Law

- Weak scalability measures how the time-to-solution for a problem varies as a function of the number of processing units when there is a *fixed problem size per processing unit*.
- Weak scalability is limited by Gustafson's Law, which states that the theoretical speedup of a parallel algorithm is given by

$$S_n = n - f_s^*(n-1),$$

where $0 \leq f_s^* \leq 1$ is the sequential fraction of the parallel execution runtime. Note: $f_s^* \neq f_s$.

- If $f_s^* \to 0$, then $S_n \to n$.



An Introduction to OpenMP

...

https://computing.llnl.gov/tutorials/openMP

hello, world (c)

```c
#include <stdio.h>

int main(void) {

    printf("hello, world\n");
    return 0;

}
```

hello, shared memory world (c)

```c
#include <stdio.h>
#include <omp.h>

int main(void) {

    int number_of_threads, thread_id;

    #pragma omp parallel private(thread_id)
    {
        thread_id = omp_get_thread_num();

        #pragma omp master
        {
            number_of_threads = omp_get_num_threads();
        }

        #pragma omp barrier
        printf("hello, shared memory world from thread %d of \
            %d\n", thread_id, number_of_threads);
    }

    return 0;

}
```

hello, world (fortran)

```
program hello

use, intrinsic :: iso_fortran_env
implicit none

write(unit=output_unit, fmt=*) "hello, world"

stop

end program hello
```

hello, shared memory world (fortran)

```
      program hello_omp

      use, intrinsic :: iso_fortran_env
      implicit none

      integer :: number_of_threads, thread_id
      integer :: omp_get_num_threads, omp_get_thread_num

!$omp parallel private(thread_id)
      thread_id = omp_get_thread_num()
!$omp master
      number_of_threads = omp_get_num_threads()
!$omp end master
!$omp barrier
      write(unit=output_unit, fmt=*) "hello, shared memory&
         & world from thread ", thread_id, " of ",&
         & number_of_threads
!$omp end parallel

      stop

      end program hello_omp
```

Matrix-Vector Multiplication

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

Matrix-Vector Multiplication

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Matrix-Vector Multiplication

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ a_{21}x_2 + a_{22}x_2 + \cdots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \end{bmatrix}$$

Matrix-Vector Multiplication

$$y_i = \sum_{j=1}^{n} a_{ij} x_j$$

where $i = 1, 2, \ldots, m$.

Hilbert Matrix

A Hilbert matrix is a square matrix with elements that are unit fractions given by

$$H_{ij} = \frac{1}{i + j - 1}$$

For example, the Hilbert matrix of dimension 4 is

$$\mathbf{H} = \begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{bmatrix}$$

Exercise: Matrix-Vector Multiplication

If $\mathbf{H}$ is a Hilbert matrix of dimension $n = 122880$ and $\mathbf{x} = [1 \cdots 1]^T$ is an all-ones vector of the same dimension, compute the resultant vector $\mathbf{y} = \mathbf{H}\mathbf{x}$.

Check your result by computing the sum of the elements of $\mathbf{y}$. For $\mathbf{H}$ and $\mathbf{x}$ of dimension n, the sum of the elements of $\mathbf{y}$ is given analytically by

$$\sum_{i=1}^{n} y_i = n + \sum_{j=1}^{n-1} \frac{n-j}{n+j}.$$

Extra credit: Optimize your code and recompute for both $n = 122880$ and $n = 1048576$. Plot the parallel speedup and efficiency of your code.

Questions?