MultiProcessing Mathematics

Or
A Good Reason to
Spend the Money and Buy
a Multicore CPU Chip

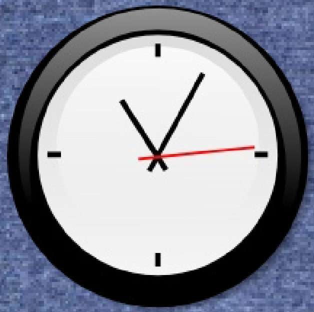
By Brad Morantz PhD

Forward

- Multiprocessing mathematics is not the focus of my research
- My research is in intelligent data analysis and decision-making utilizing advanced methodologies and machine-cognition
- Being able to process the numbers quicker and in parallel is necessary for my work

Computationally Intensive

- Some problems take forever to calculate
  - TSP Travelling Salesman Problem
  - Weather forecast modeling
  - Large simulations
  - Many more
- Sometimes takes too long
- Clock speeds are about as fast as they can get
- Are you going to wait for a Feynman quantum calculating chip? How long for that???



On the Other Hand

There are many multicore processors on the market today

Six core, eight core, hyper threaded
More every day

GPU

- GPU = Graphics processing unit
- Limited functionality
- Some chips have as many as 256 cores
- Network of 6 PS3 playstations
- Nvidia CUDA GPU language
- Nvidia Tesla desktop supercomputer
  - 515 GFLOPS to over 4 PetaFLOPS
- Big processing power at small price

What Gain?

- So, is your math program using more than one of the cores? Probably not!

- Look at your core usage while the program runs.
- The above picture is the goal, all of the CPUs in use.

Goals

- Solve the problem clearly and efficiently
- Focus on ***the problem*** not the code
- Generate code that:
  - Runs fast
  - Is Parallel processing code
  - Is easy to understand
  - Written in math
  - Comprehendable by mathematicians and SME (subject matter experts)
  - Will be understood in six months
  - Is standard and portable
  - Can be taken anywhere
  - Will run on any platform
  - That is close to the mathematical formulation
  - Code that is clear and concise

Most Important

- Why bother to write a program if it does not produce the right answer
- Or maybe sometimes right, sometimes wrong
- Enter Verification and Validation
  - Known as V&V (or IV&V)
  - Subject for a whole semester
  - Mandatory

Problems

2 + 2 = 5
5 X 3 = 14.7
12 / 3 = 6
10 – 4 = 5

Verification

- Boehm: "Are we building the product right?"
- Does the software correctly implement

  our function?
- Check the envelope for consistency
- The more variables, the more likely to have a place where the system "goes postal"
- Design Analysis Simulation Experiments
  - Full semester to study this
    - Taguche, Latin Hypercube, Exploited search

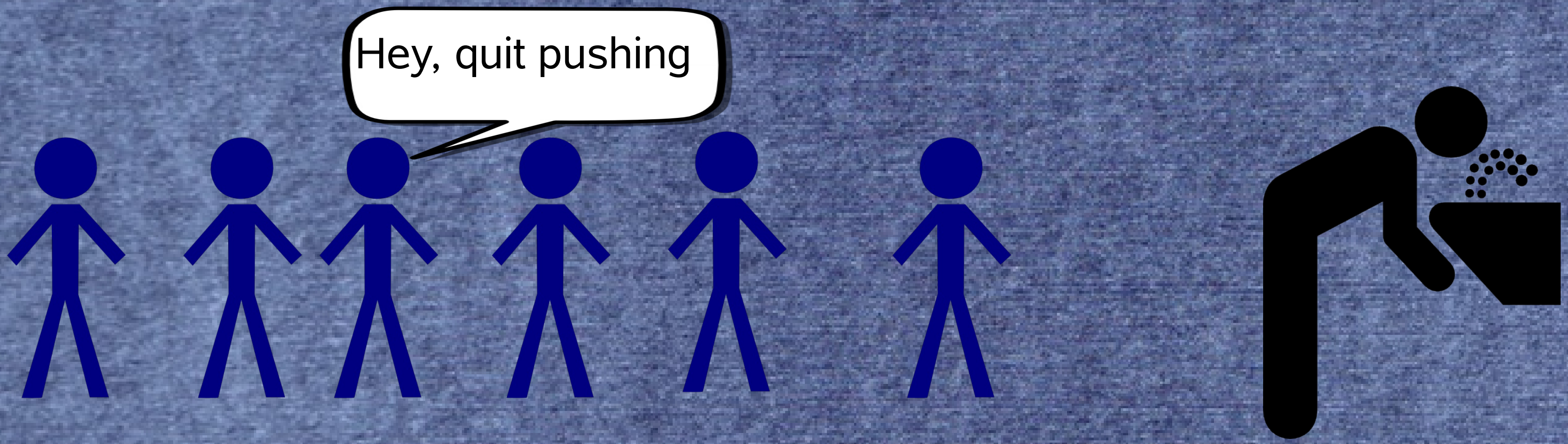
Validation

- Boehm: "Are we building the right product?"
- In our situation, does this program really solve our math problem?
- Do not want Type III error
  - Right answer to the wrong problem
- Get a mathematician to check it over
  - Must be able to comprehend the code

Definitions

- SISD
  - Single Instruction Single Data
- SIMD
  - Single Instruction Multiple Data
- MIMD
  - Multiple Instruction Multiple Data

Typical Serial Processing



It can take a while until everyone has had a drink

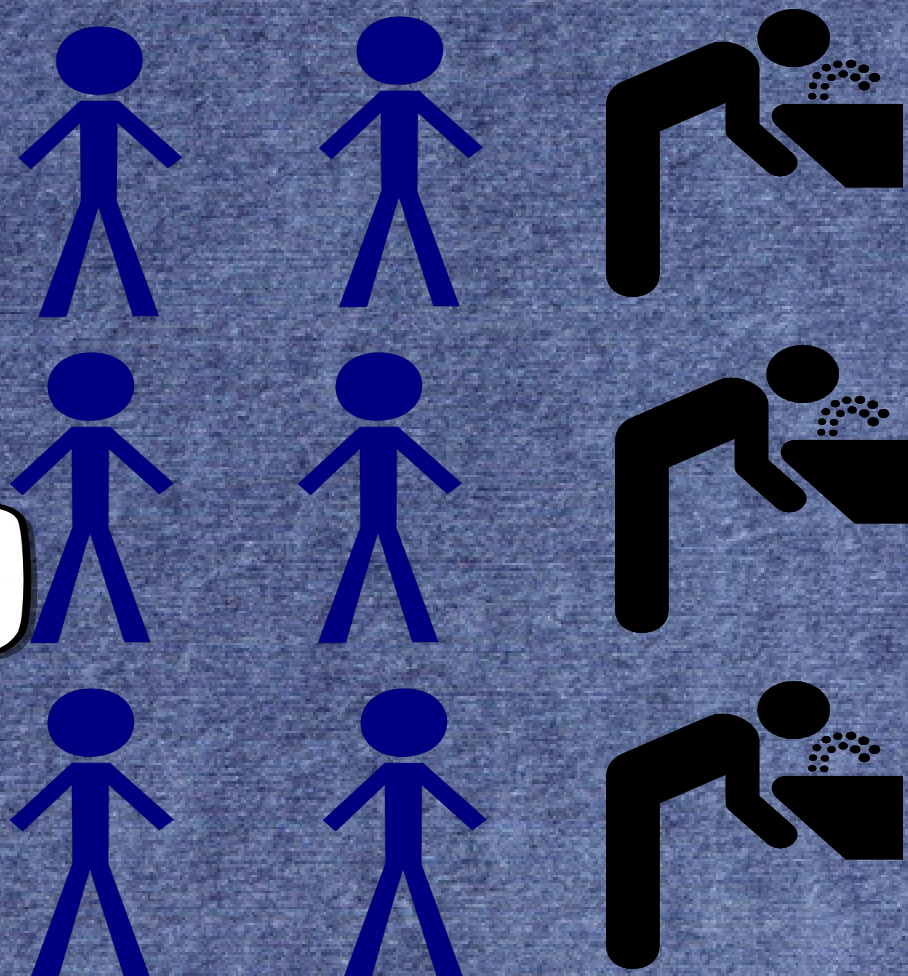
Parallel Processing

Multiple Drinking Fountains

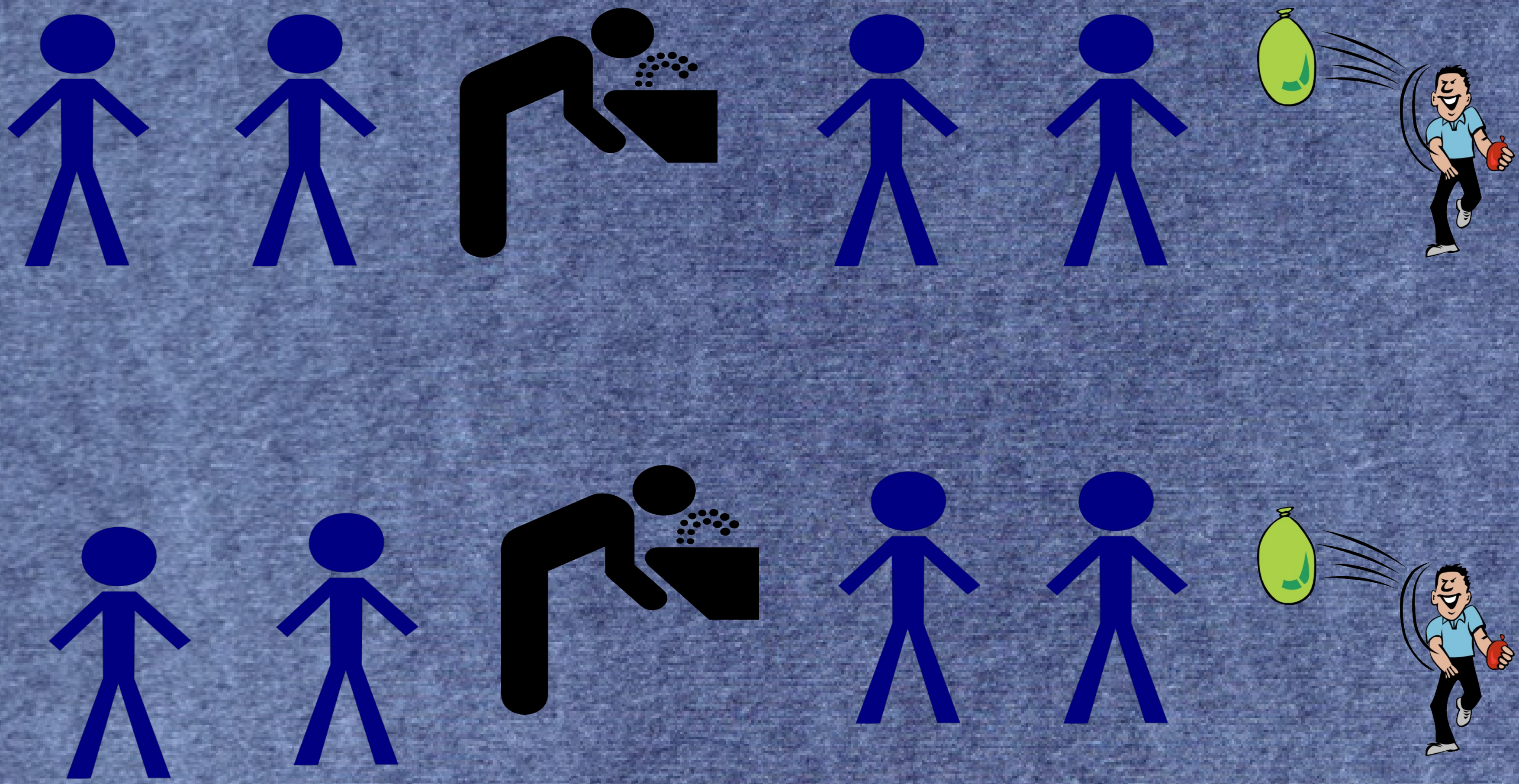
This will process three
Times as fast

Leave some for the fish!

Queuing would be even
Faster if the processes
Were of varying length
(Someone very thirsty)



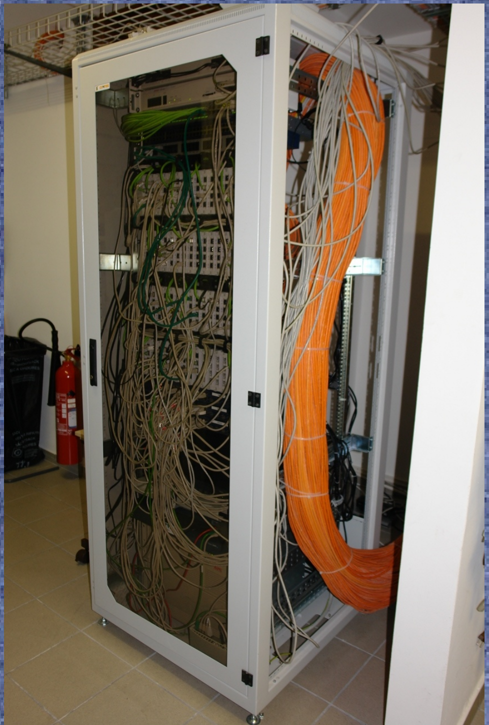
MIMD Processing



Two different processes going on at the same time

Non Equal Length Tasks

- Non trivial problem
- Best to use a modeling & simulation package
- Model each task and the processors
- Optimize for maximum throughput
- My experience
  - 72 processor Sun box
  - Complicated process
  - Modeled using Arena
  - Result: double throughput



Some Typical Applications

- Testing, system verification
- Genetic Algorithms
- Neural networks
- Structural equation modeling
- Large simulations
- Cluster analysis
- Statistical analysis of large data sets
- Signal processing of complex waveforms

Tight vs Loose Structure

- Some languages automatically adjust vector or matrix size to fit the situation
  - Does not tell when there is a problem
  - Programmer is not in control
- Some languages require exact definitions
  - Running over dimensions or size gives an error message or warning
  - This alert can save many problems
  - Requires careful work of programmer

Example Problem

- I want to declare some matrices, fill them with values calculated in long equations, and then multiply them.
- And then test this program on various machines and with varying number of processors.
- I did this and got interesting results (see next page)

Results

- For 1000 x 1000 matrix
  - On my old Athlon 2000 (Linux)
    - 142 seconds
  - My dual Athlon 1800 (Linux)
    - One processor 152 seconds
    - Two processors 76.5 seconds
  - My wife's Athlon 3000-64 (Linux)
    - 5 seconds
  - Old Job on Dual Quad Core Xeon (Linux)
    - One processor 2.8 seconds
    - Eight processors 0.24 seconds
  - My Hyperthreaded HP laptop at RMS (Core duo Windows)
    - 126.25 seconds

The code

```
Program Tryitout
Implicit none
real*16, dimension(:,:), allocatable        :: A, B, C   ! 16 byte floating point matrix, 2D
integer*4                                    :: row, i, j
real*4                                       :: starttime, donetime
print*, 'What size array? Start with square array'
read*, row
allocate(A(row,row), B(row,row), C(row,row))
call cpu_time(starttime)
blockit: forall (i = 1:row:1, j = 1:row:1)
A(i, j) = ((real(i)) **2) * sin(real(i)/real(j))
B(i, j) = (real(i))/(real(j)) * cos (real(i/j))
end forall blockit
c = matmul(a,b)
call cpu_time(donetime)
print*, 'it took ',(donetime-starttime), ' seconds'
end program tryitout
```

Some Comments

- Want to be able to use complex variables
  - No change in program
  - Except variable declaration
- Want to be able to control precision
  - With large number of iterations
- Want to put it across all processors
  - Run faster
- Start with a plan, a design
  - The best made things have a design first
  - Plans are mandatory for good design

Think Parallel
(The Hardest Part)

- Programmer must think parallel
  - Think matrices not scalars
  - Think parallel not serial
  - Think wide not narrow
- Algorithm must be parallel
- Think in parallel mode
  - Will this go into a matrix
  - How can I put this into a vector
  - How can this be made more parallel
- How can this be done in parallel mode

Parallel Algorithm

- Work out algorithm
- Draw pictures and diagrams
- Work for parallel processes
- Make flow chart
- How can things go into loops or matrices
- Independence (next slide)

Independence

- Matrices and loops can be automatically parallelized
- Must be independent
  - One value in array can not be dependent upon result of another
  - The order must not be important
- Think if you had a large problem
  - Have friends helping you
  - Give each of them part of problem to do

Where is the Parallelization

- The old way (1 CPU or core)
  - Do 100 J = 1, 1000, 1
  - X(J) =J**2
  - 100 continue
- The Parallel way (using multiple CPUs or cores)
  - Forall (J = 1:1000:1) X(J) = J**2
- So what is really happening?
  - The compiler first checks for independence
  - Then it divides the 1000 by the number of CPUs
  - Then it puts the process across all of the CPUs
    - Process 1 to 250 on CPU #1
    - Process 251 to 500 on CPU #2
    - And so on

Parallelization Considerations

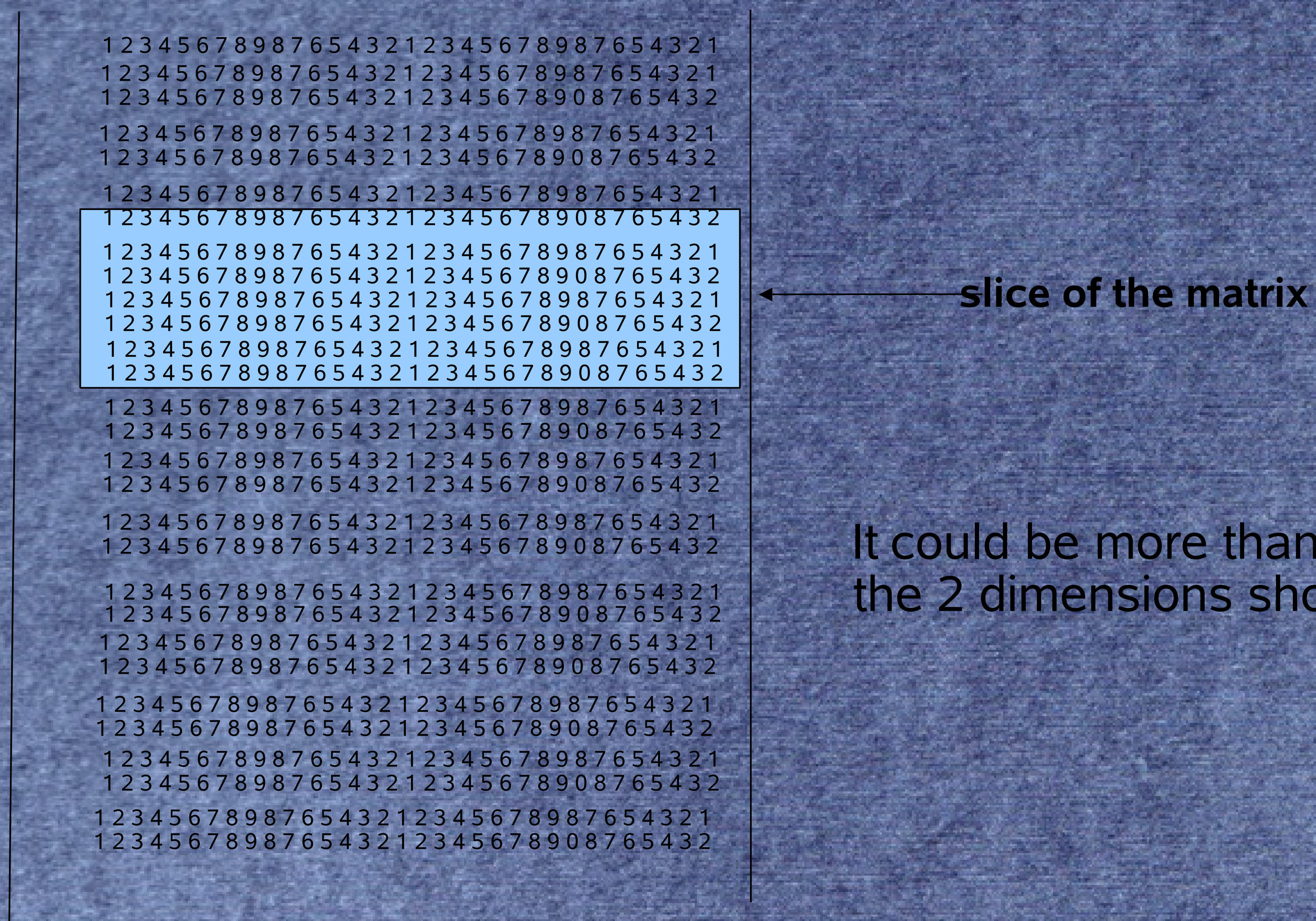
- Overhead
  - Setting up and supervising takes work & time
  - Usually not worth it for small number iterations
  - Can exceed savings if not careful
- Control
  - Some compilers offer control
  - Set integer
    - Usually from 1 to 100
    - Do not parallelize if under this number

Another Example

Suppose:

- You had a matrix of numbers (a whole bunch of numbers) and wanted a slice of it
- And there were some missing numbers (signified by the value 9999.0)
- And you wanted to get the mean of columns in this slice, creating a new matrix of 1 less dimension
- And you wanted to utilize all of your processors.
- And you want the code to be maintainable, easy to understand, & portable

Matrix



It could be more than
the 2 dimensions shown here

Matrix Code

```
mean(row,:) = sum(inarray(first:last,:), dim=1,
   mask=inarray(first:last,:) .NE. 9999.0)
! note: arrays must be conformable
adjust(:) = count(inarray(first:last,:) .EQ. 9999.0, dim=1)
mean(row, :) = mean(row,:)/(last- first +1 – adjust(:))
!  this is code out of a program
```

- **This does it all in these 3 lines.**
- And it uses **all of the processors**.
- It is even doing it on slices out of a big matrix. It creates a matrix of means.
- Actual working code out of one of my programs
- Could have been in 1 line, but would have been hard to read

Mathematical Languages

- Matlab, Mathematica, Octave, Scilab
  - Interpretive
  - Matlab & Mathematica have some multiprocessing functions
    - Costs $$$
  - Octave & Scilab are free
- Fortran 95/2003/2008
  - Do NOT confuse with the old Fortran 77
  - Compiled
  - Many multiprocessing implementations
  - Many free compilers
  - Comparisons at www.polyhedron.com
- Many more

Graphics Programming Languages

- For Graphic Processing Units
- Yorick
  - Scripting matrix language (Interprative)
  - Created by a physicist
  - C like syntax
  - Free for Linux
- Nvidia CUDA

Mathematica

- New in Mathematica 7
  - Parallelize - for matrices and vectors
  - ParallelTry – Tries a function in parallel
- Multicore parallelism standard
  - with zero configuration
- Flexible data parallelism functions built-in
- Built-in interface for GPU computing
- GridMathematica (more $) is option
  - Can run as much as 4 tasks in parallel
- Mathematica 8 uses CUDA

Portland Group

- Workstation x64
  - Fortran 2003 optimizing multicore compiler
- CUDA Fortran
  - GPU acceleration in native optimizing compiler

Intel

- Fortran Composer XE 2011
  - Optimizing
  - Multiprocessing
  - Math Kernel Library
  - Free academic use only for Linux

Absoft

- Multiprocessing
- GPU support
- Optimizing
- Fastest from Polyhedron tests
- Expensive

GNU Free Software Foundation

- Free! (no charge)
- Will parallelize with OMP
  - Not as easy
  - Extra work
- Code must be exact per specs

Overview Fortran 95/2003/2008

- Automatic Parallelization
- Matrix functions
- Object oriented
- Complex math built in
- 16 byte floating point (32 byte complex)
- Bit and string functions
- Structures and arrays
- ISO standard
- Backward compatibility
- Dynamic allocation
- From matlab code to Fortran in minutes
- Originally started as HPF at Rice/MIT

Parallel Functions

- ALL – are all values in mask true?
- ANY – are any values in mask true?
- COUNT – dimensional reduction, mask ok
- MAXVAL – reduction or scalar, mask OK
- MINVAL – reduction or scalar, mask OK
- PRODUCT – of element of array along DIM
- SUM – reduction or scalar, mask OK
- MATMUL - matrix multiply
- MERGE- merge 2 array with mask
- SPREAD (Source, Dim, Ncopies)

More Parallel Functions

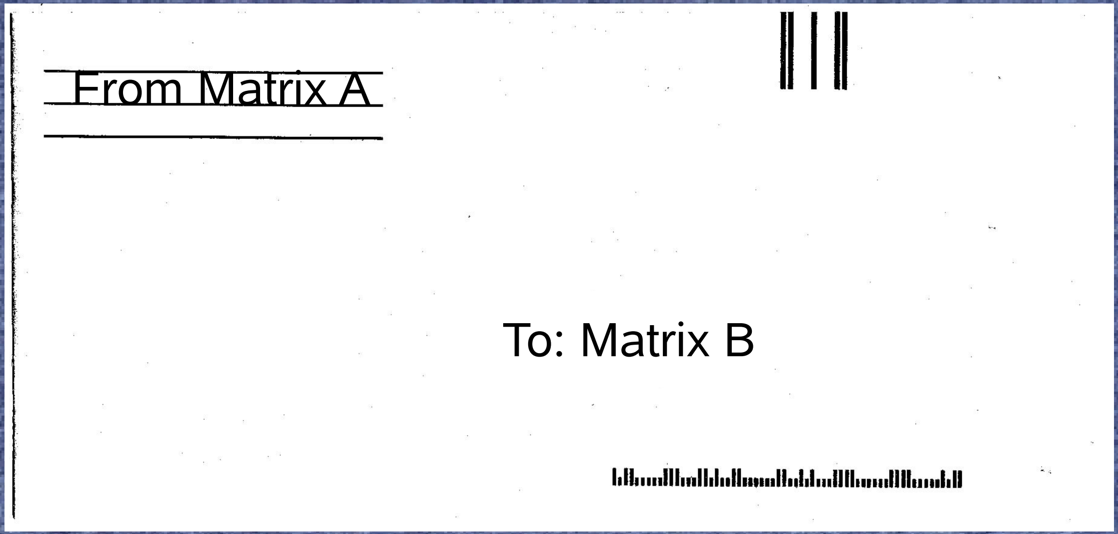
- PACK(Array, Mask [,Vector])
- UNPACK(Vector, Mask, Field)
- TRANSPOSE
- CSHIFT -circular shift
- EOSHIFT – end off shift
- MAXLOC – first location of max value
- MINLOC – first location of min value
- FORALL parallel looping
- WHERE masking function
- ELSEWHERE masking function

More Array Functions

- LBOUND - inquiry
- UBOUND - inquiry
- SIZE - inquiry
- SHAPE - inquiry
- RESHAPE - transformational
- DOT_PRODUCT- dot product multiplication
- ALLOCATE – assign storage
- ALLOCATED- inquiry

Addressing Matrices

- Address a row
  - A(row,:)
- Address a column
  - A(:, col)
- Address a slice (This is neat!)
  - A(d:f, h:12)
- Address a matrix
  - A
  - print*, A



Elemental Matrix Math

- A, B, C are matrices, same size
- C = A*B   multiplication
- C = A+B   addition
- C = A/B   division
- C = A-B   subtraction
- C = 0.0   initialization
- Automatic multiprocessing
  - If enabled
  - Depending on compiler and machine

Matrix Multiplication

- A,B,C are matrices
- C = matmul(A,B)
- Arrays must be of same type
  - integer, real, complex, or logical
- [n,m] x [m,k] = [n,k]
- [m] x [m,k] = [k]
- Automatic multiprocessing
  - depending on compiler and machine
  - if enabled

$$\Pi = \begin{pmatrix} 1 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 1 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 1 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 1 & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 1 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & \frac{1}{2} & \frac{1}{2} & 1 \end{pmatrix} \text{ with } \mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{pmatrix} = \begin{pmatrix} Q_1 + Q_2 \\ Q_3 + Q_4 \\ Q_5 + Q_6 \\ Q_7 + Q_8 \\ Q_1 + Q_4 \\ Q_6 + Q_8 \\ Q_4 + Q_6 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 3 & 5 \\ 7 & 9 & 11 \\ 13 & 15 & 17 \end{pmatrix} + \begin{pmatrix} 0 & 2 & 4 \\ 6 & 8 & 10 \\ 12 & 14 & 16 \end{pmatrix} = \begin{pmatrix} 1+0 & 3+2 & 5+4 \\ 7+6 & 9+8 & 11+10 \\ 13+12 & 15+14 & 17+16 \end{pmatrix}$$

$$= \begin{pmatrix} 0+1 & 2+3 & 4+5 \\ 6+7 & 8+9 & 10+11 \\ 12+13 & 14+15 & 16+17 \end{pmatrix} = \begin{pmatrix} 0 & 2 & 4 \\ 6 & 8 & 10 \\ 12 & 14 & 16 \end{pmatrix} + \begin{pmatrix} 1 & 3 & 5 \\ 7 & 9 & 11 \\ 13 & 15 & 17 \end{pmatrix}$$

MAXVAL

- MAXVAL(ARRAY, DIM, MASK)
- creates a new array or assigns
    - 1 less dimension
    - max value along dimension DIM
    - corresponding to TRUE elements of MASK
- Ex: maxivals = MAXVAL(A, DIM=2, MASK=A.GE.0.0)
- MINVAL is the same, but gets the minimum

MINLOC

- MINLOC(ARRAY, DIM, MASK)
- Creates or assigns array of 1 less dimension
- Finds location of first element of ARRAY having minimum value of elements as defined by MASK
- DIM is optional
- Ex: MINLOC(A, DIM=1, A.GT.12)

FORALL

- Like the do loop, but uses all available CPUs
- FORALL(I=1:N, J=1:M) A(I,J) = I+J
- Loop1: FORALL (I=1:N:0.50)

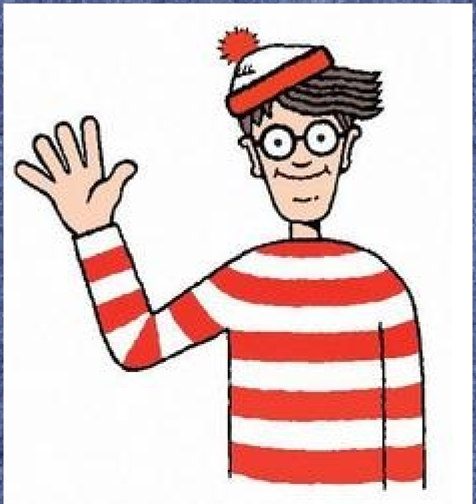
WHERE (A(2*I).GE.43), A(INT(I)) = I**2

end loop1

(Naming loops is optional, but when used, compiler will match names. Another case where time spent in organization pays off in accuracy and performance)

- Part of HPF specification (Rice/MIT)
  - automatic multiprocessing

WHERE



- Masked array assignment

hot: WHERE(A >= 27.32)
temp = temp + degree
ELSEWHERE
cold = cold + degree
END WHERE hot

- WHERE (A < 0.0) A = -A
  - Single line, creates non-negative matrix
  - A could be N dimension matrix
  - Can use multiprocessors
  - No need for indices

Assignments

- A, B same size matrices
  - A = B
  - A(f:g, 3) = B(f:g, 5)
  - c = A(x,y) * B(p,q)
  - D = A(1:3, 2:4) * B(23:25, 16:18)
- Automatic multiprocessing
  - If enabled
- Target must be same shape and size as source
- Or source can be a scalar

Other Mathematical Features

- 128 bit (16 byte) variables/ 32 byte complex
- Compiler takes care of the work
- Complex math (on all complex numbers)
- Arrays of structures
- Structures of arrays (including allocatable)
- RECURSIVE
- Operator & function overloading
- Modules
- PURE
- Floating point exception handling
- IEEE floating point math

Math Library

- The Intel Fortran comes with a complete library
  - Optimized for multiprocessing
  - Based on IMSL & BLAS
- Others are not multiprocessing
  - GNU Scientific Library
  - Lapack
  - Many more
  - With source code can be parallelized
- Math Library available with others for extra $

Summary

- Fortran 95/2003/2008
  - is for computationally intensive applications
  - Provides fast processing
  - Inherent parallelism
  - Compiles to assembly code
  - Runs efficiently
  - Converts quickly and easily from Matlab etc.
  - Deliverable tool
  - Multiprocessing with little effort

Multi Language

- Can use languages together
- Computation in Fortran and GUI in C or Java
  - Wrapper
  - Very common
- Octave, Scilab, & R can have functions in C/C++ or Fortran
  - Also common
  - Can be multiprocessing functions

The Process

- Design and develop parallel algorithm
- Test out algorithm
- Make flowchart
- Develop the algorithm in Matlab or Mathematica
- Then put it into Fortran 95/2003/2008
- Do not parallel where overhead > savings
- V&V
  - Mathematician to validate algorithm
  - Test matrix and verify output

Resources

- www.g95.org
  - free compiler, no multi-processing
    - promises multiprocessing in future
  - lots of links
- Intel (lots of info on optimization)
  - www.intel.com/software/products/compilers
  - Free compiler for home use only (Linux)
  - With automatic multi-processing (APO)
- www.fortran.com
- www.gnu.org
  - Free compiler GCC - 4.1+
  - Gedit – free editor, color coded
- www.mhpcc.edu/training/tutorials

Intel Notes

It is Sometimes easier for the Fortan compiler to optimize matrix code than for the C compiler because the C standard permits more hidden aliasing of pointers than does Fortran

Neural Net

- How many LOC's in C/C++?
- Wait until you see it in Fortran 95/2003
  - 4 lines!!
  - Maintainable
  - Mathematical
  - Easy to understand

Function linear . . . out = matmul(A,B)

Function logistic . .(sigmoidal or hyperbolic tan)

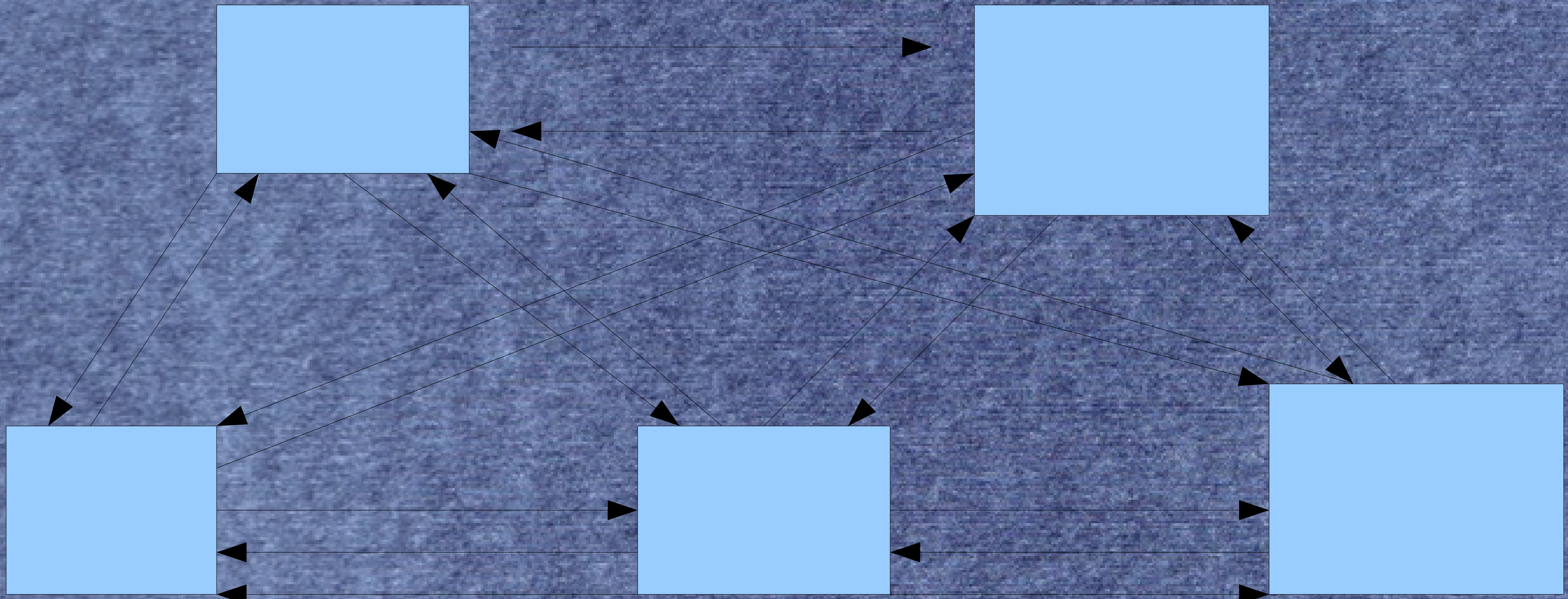
Mid = logistic(matmul(input, weight1))

Out = linear(matmul(mid, weight2))

Structural Equations

- Currently modeled on single processor boxes
  - Lisrel, PLS, etc
- Complex system where sections affect each other
- Large number of processors can perform this without trying to model and simulate
  - More accurate answer
  - See response over time
  - Can see effect of connections
- Specialized graduate course

Simple S. E. System



My Philosophy

As a decision scientist, my goal is to solve problems and find the best possible solution. The best answers are based upon knowledge and information.

Contact Information

- Brad Morantz
- Web page: www.machine-cognition.com
- Personal e-mail bradscientist@ieee.org

Questions?