Scenes from a Graphical, Parallel Biological World

Parallel graphics hardware made by nVIDIA (http://www.nvidia.com)

Programming using CUDA (compute unified device architecture)

Emerging approach to analysis and simulation for a variety of problems

I am interested in finding novel representational paradigms for biological problems, not just “speeding things up”.

Bradly Alicea

Embryo Physics Course, April 2012

http://www.msu.edu/~aliceabr/

What’s so special about a graphical, parallel approach to computational biology?

G

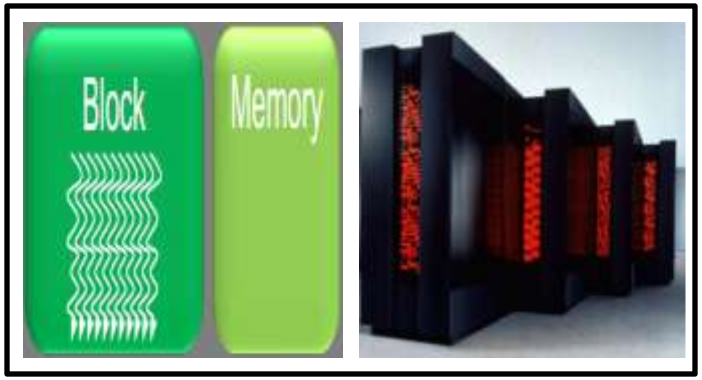
GRAPHICS: hardware designed for rendering computer graphics – video games, movie special effects.

P

PROCESSING: data handling strategies designed for parallel computation.

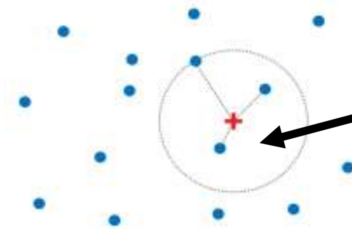
U

UNIT: block that resembles a microprocessor, often arrayed in parallel.

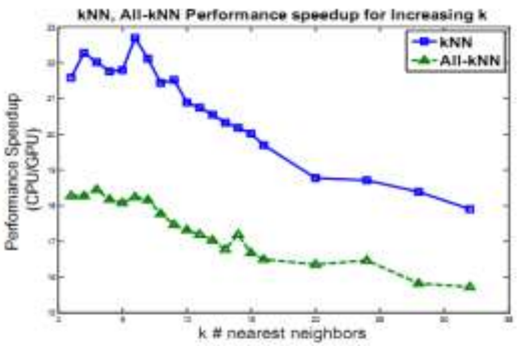


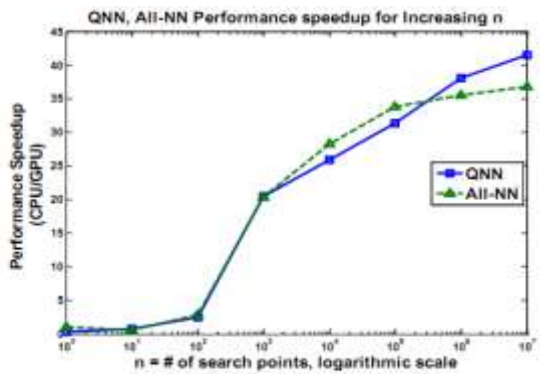
Comparing parallel GPUs to parallel CPUs

Speedup for certain tasks (brute-force kNN search) – due to more on-chip memory.



Fast k Nearest Neighbor Search using GPU, arXiv:0804.1448.

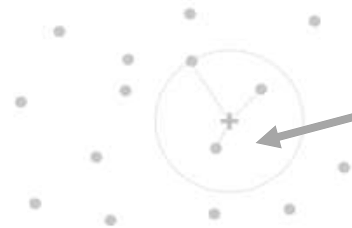




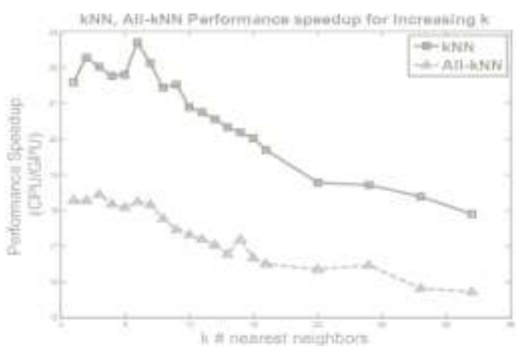
Brown and Snoeyink, GPU Nearest-Neighbor Searches using a Minimal kd-tree

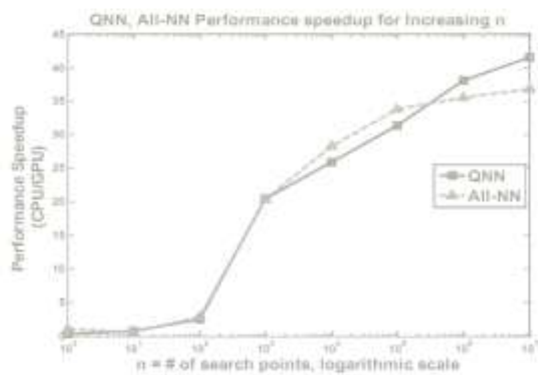
Comparing parallel GPU to parallel CPU computing

Speedup for certain tasks (brute-force kNN search) – due to more on-chip memory.

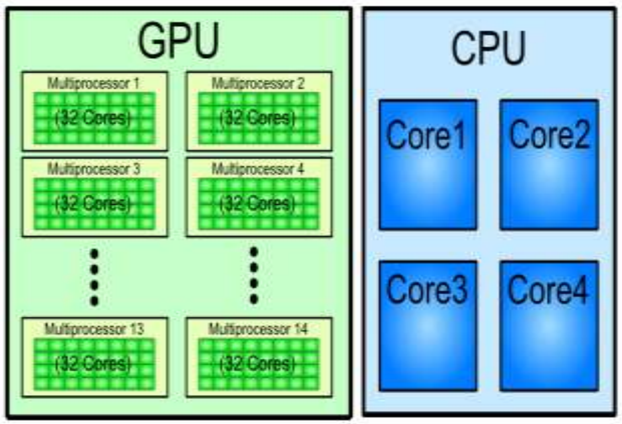


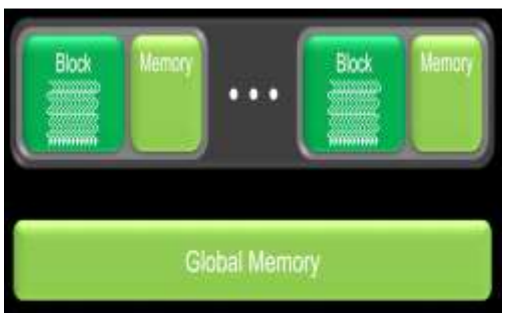
Fast k Nearest Neighbor Search using GPU, arXiv:0804.1448.





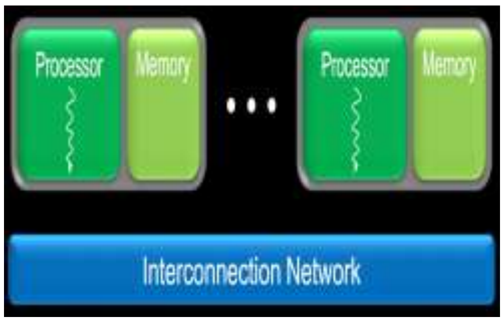
Brown and Snoeyink, GPU Nearest-Neighbor Searches using a Minimal kd-tree





GPU Parallel Processor Design

* multithreaded processor (multiple blocks per core, multiple cores).



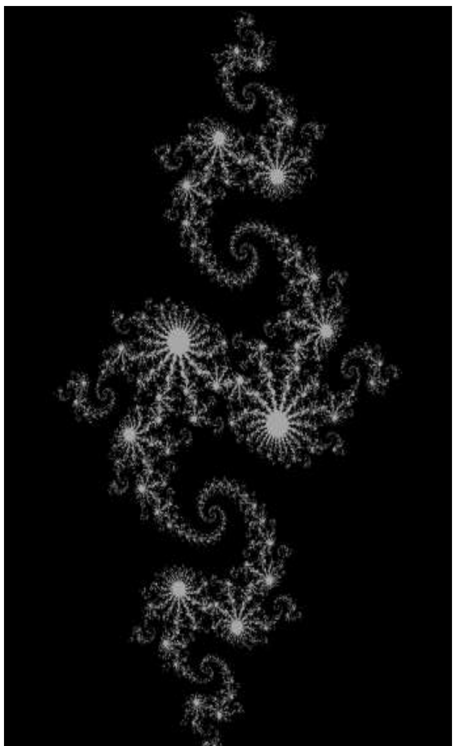
CPU Parallel Processor Design

* no global (shared) memory, multiple cores.

Julia Set – CUDA for CPU

Julia Set – CUDA for GPU

Map complex input data to a grid, block structure



```
int main( void ) {
   CPUBitmap bitmap( DIM, DIM );
   unsigned char *ptr = bitmap.get_ptr();
   kernel( ptr );
   bitmap.display_and_exit();
}
void kernel( unsigned char *ptr ){
   for (int y=0; y<DIM; y++) {
      for (int x=0; x<DIM; x++) {
         int offset = x + y * DIM;
         int juliaValue = julia( x, y );
         ptr[offset*4 + 0] = 255 * juliaValue;
         ptr[offset*4 + 1] = 0;
         ptr[offset*4 + 2] = 0;
         ptr[offset*4 + 3] = 255;
      }
   }
 }
int julia( int x, int y ) {
   const float scale = 1.5;
   float jx = scale * (float)(DIM/2 - x)/(DIM/2);
   float jy = scale * (float)(DIM/2 - y)/(DIM/2);
   cuComplex c(-0.8, 0.156);
   cuComplex a(jx, jy);
   int i = 0;
   for (i=0; i<200; i++) {
      a = a * a + c;
      if (a.magnitude2() > 1000)
         return 0;
   }
   return 1
}
```

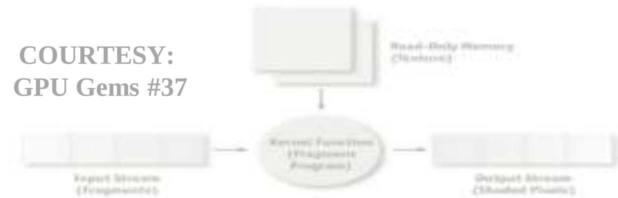
```
int main( void ) {
   CPUBitmap bitmap( DIM, DIM );
   unsigned char    *dev_bitmap;
   HANDLE_ERROR( cudaMalloc( (void**)&dev_bitmap,
                      bitmap.image_size() ) );
   dim3    grid(DIM,DIM);
   kernel<<<grid,1>>>( dev_bitmap );
   HANDLE_ERROR( cudaMemcpy( bitmap.get_ptr(),
                      dev_bitmap,
                      bitmap.image_size(),
                      cudaMemcpyDeviceToHost ) );
   bitmap.display_and_exit();
   cudaFree( dev_bitmap );
}

HANDLE_ERROR( cudaMemcpy( bitmap.get_ptr(),
                      dev_bitmap,
                      bitmap.image_size(),
                      cudaMemcpyDeviceToHost ) );

__global__ void kernel( unsigned char *ptr ) {
   // map from threadIdx/BlockIdx to pixel position
   int x = blockIdx.x;
   int y = blockIdx.y;
   int offset = x + y * gridDim.x;
   // now calculate the value at that position
   int juliaValue = julia( x, y );
   ptr[offset*4 + 0] = 255 * juliaValue;
   ptr[offset*4 + 1] = 0;
   ptr[offset*4 + 2] = 0;
   ptr[offset*4 + 3] = 255;
}
```

COURTESY: CUDA SDK, nVIDIA forums (forums.nvidia.com)

Traditional approach: execute a program or algorithm on a single processor (serially).

Parallel approach: execute a program or algorithm across multiple processors simultaneously.

GPU uses stream processing



* execute a function on a set of input data, produce a set of outputs (operations done in parallel).

* no dependencies between elements.

Traditional approach: execute a program or algorithm on a single processor (serially).

Parallel approach: execute a program or algorithm across multiple processors simultaneously.

Parallel ≠ Serial:

1) P-completeness: **P ≠ NC**

P = class of problems solvable in polynomial sequential time.

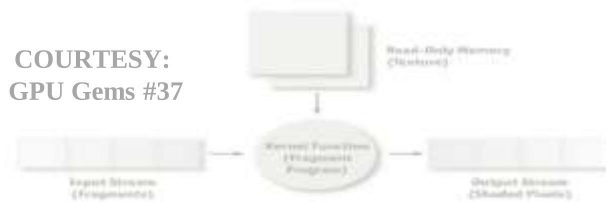
NC = class of problems solvable in polyalgorithmic parallel time using n parallel processors.

Additional cores ≠ linear scaling of performance:

2) P $[2^{O(\log\ n)}]$ ≠ **NC** $[2^{O(\log\ n)}$ using O(n) processors]

GPU uses stream processing

COURTESY:
GPU Gems #37

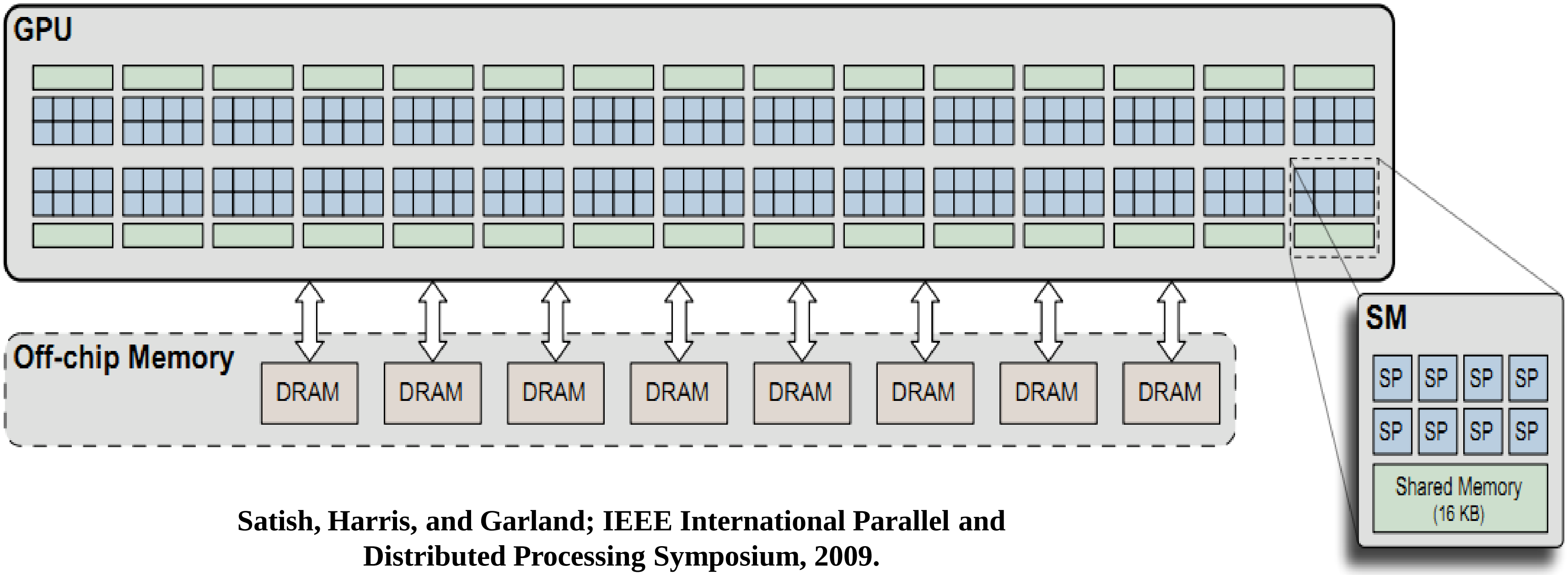


* execute a function on a set of input data, produce a set of outputs (operations done in parallel).

* no dependencies between elements.

Single core = 1x	**Four cores = 3.16x**

Graphical, parallel biology *in silico* (on computing hardware)



Satish, Harris, and Garland; IEEE International Parallel and Distributed Processing Symposium, 2009.

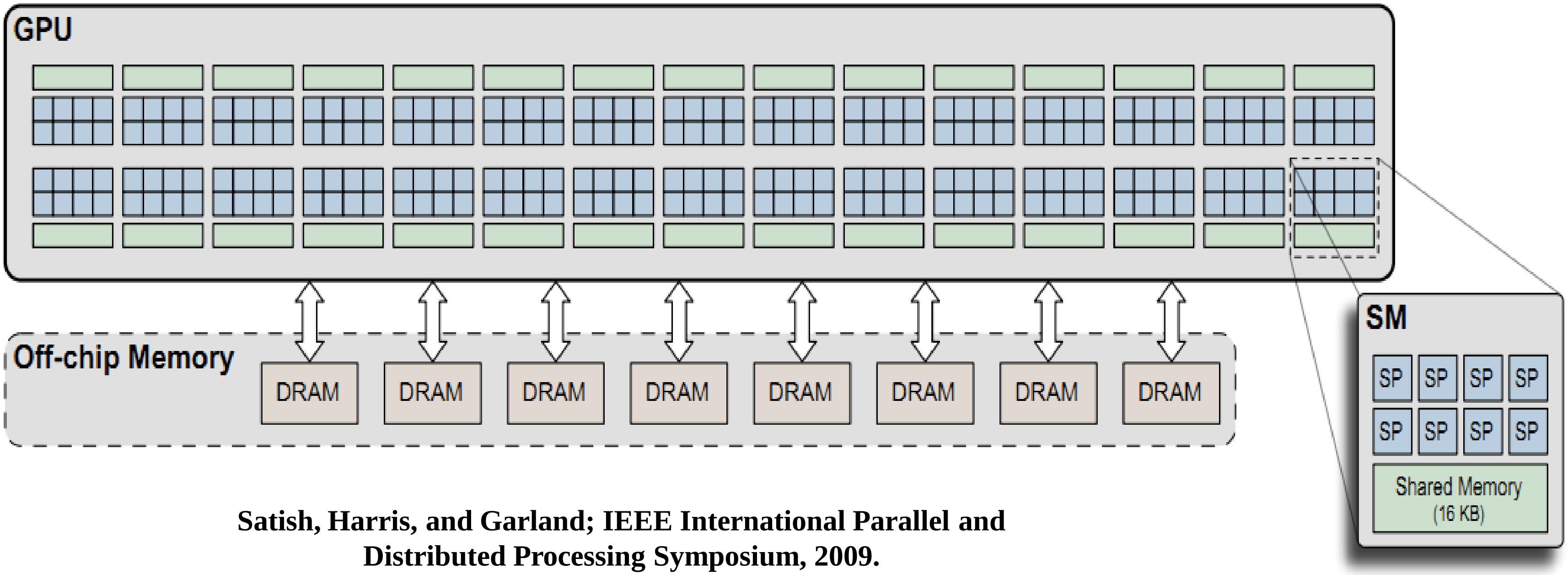
GPU kernel (function):

* loads data and defines data structures.

* defines operations on data (stream processing).

* maps to a thread, block structure.

Graphical, parallel biology *in silico* (on computing hardware)



Satish, Harris, and Garland; IEEE International Parallel and Distributed Processing Symposium, 2009.

GPU kernel (function):

* loads data and defines data structures.

* defines operations on data (stream processing).

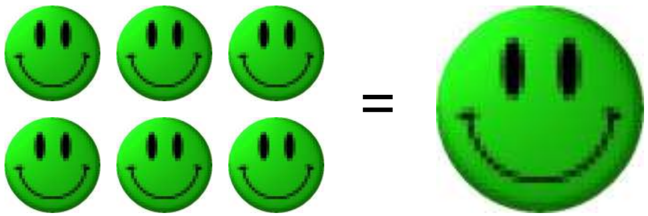
* maps to a thread, block structure.

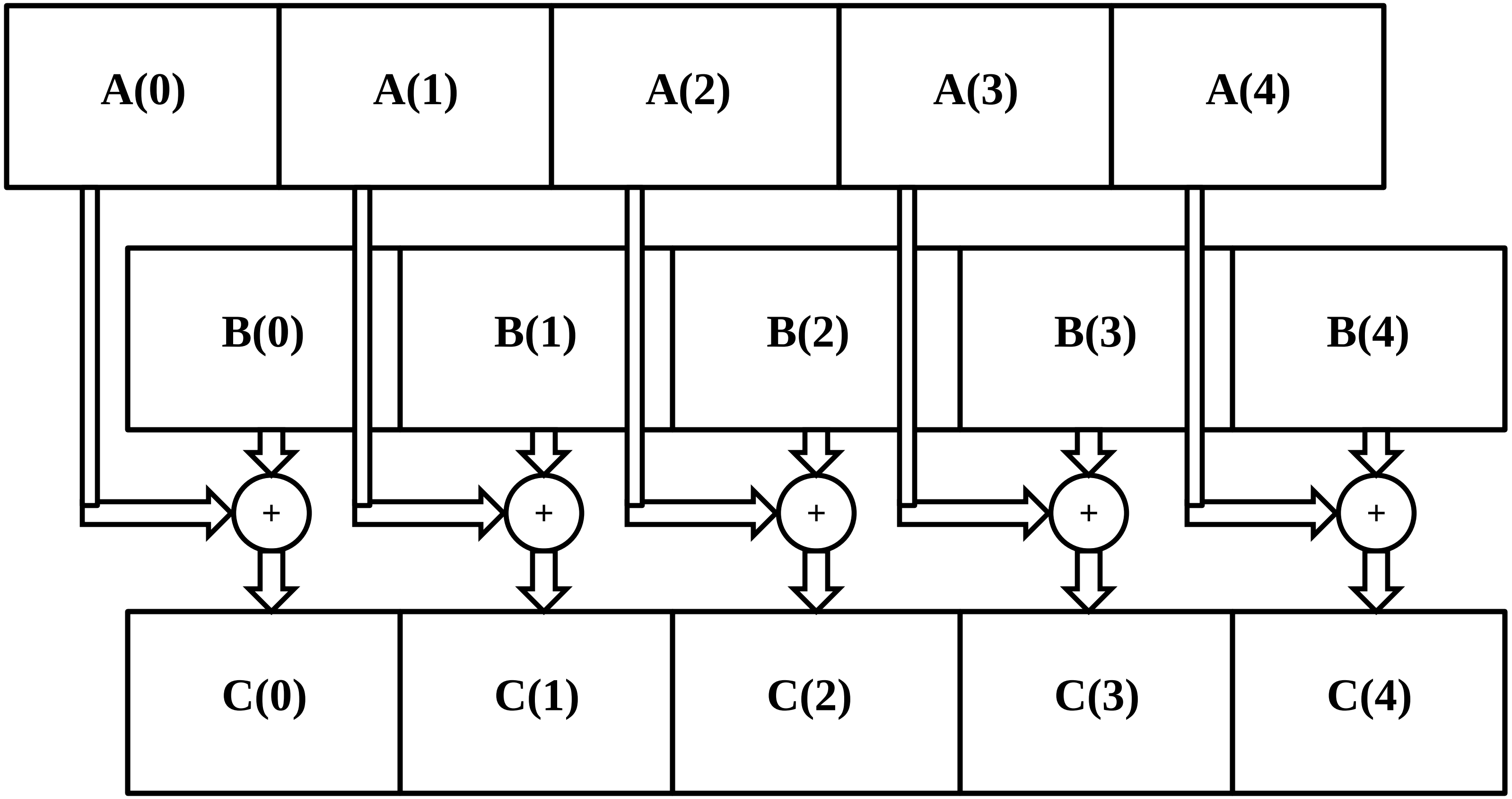
Special features of GPU computing:

* fast on-chip memory, off-chip RAM is separate (CUDA architecture).

* distribute problem to multiple threads, iterate, reassemble in global memory.

A typical "Hello World" example for parallel, graphical computing

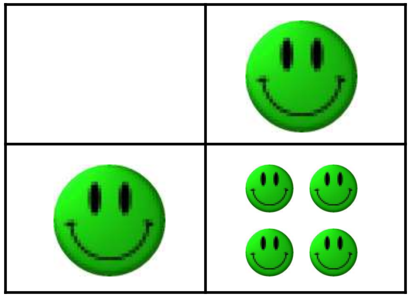


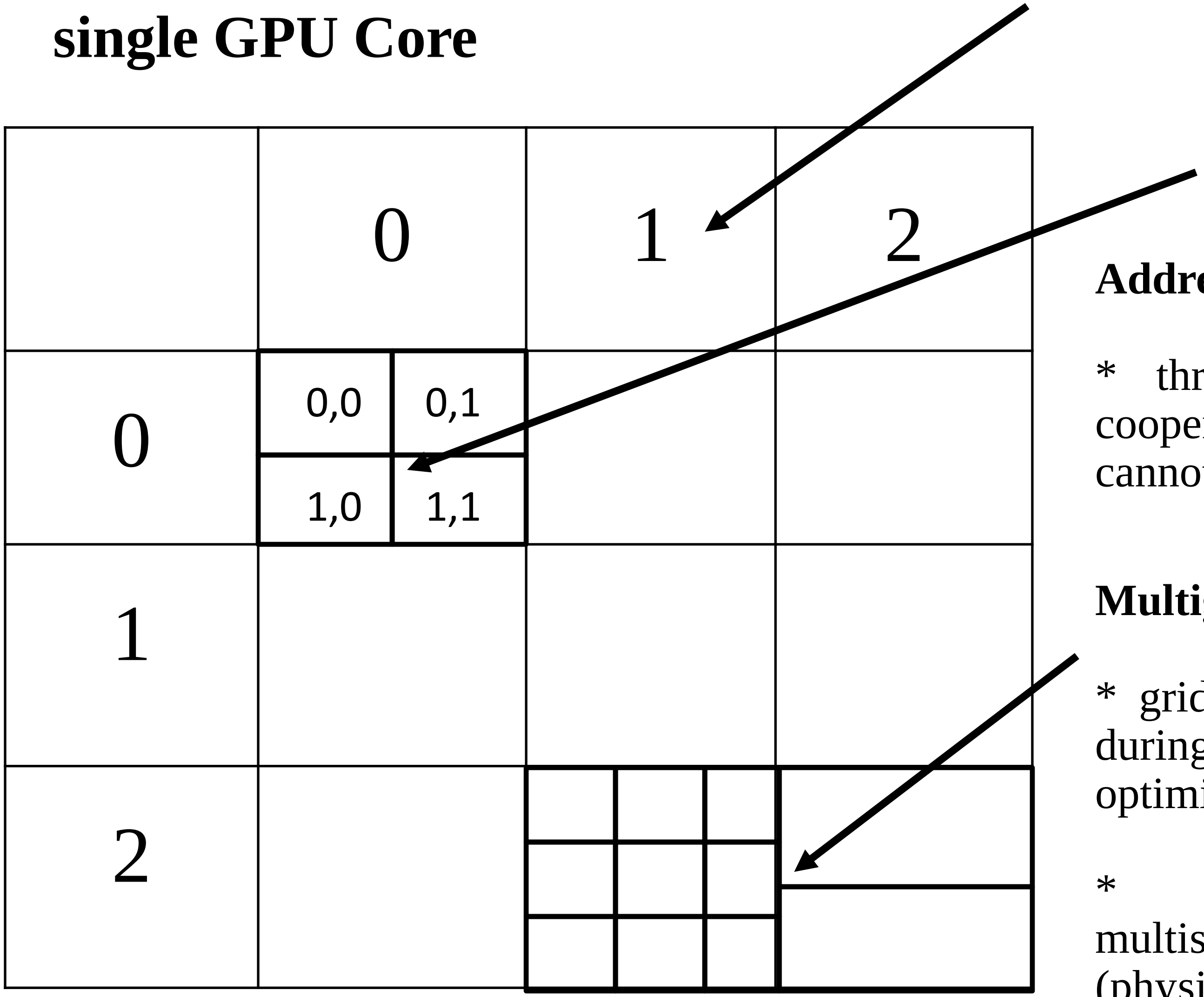


Vector Addition

1) Read element-wise (allocate memory for A,B).
2) Launch kernel, device performs vector addition.
3) C copied from device memory after finished.

Representing Block and Thread Indices on a single GPU Core





Address system:

* thread elements can cooperate, block elements cannot.

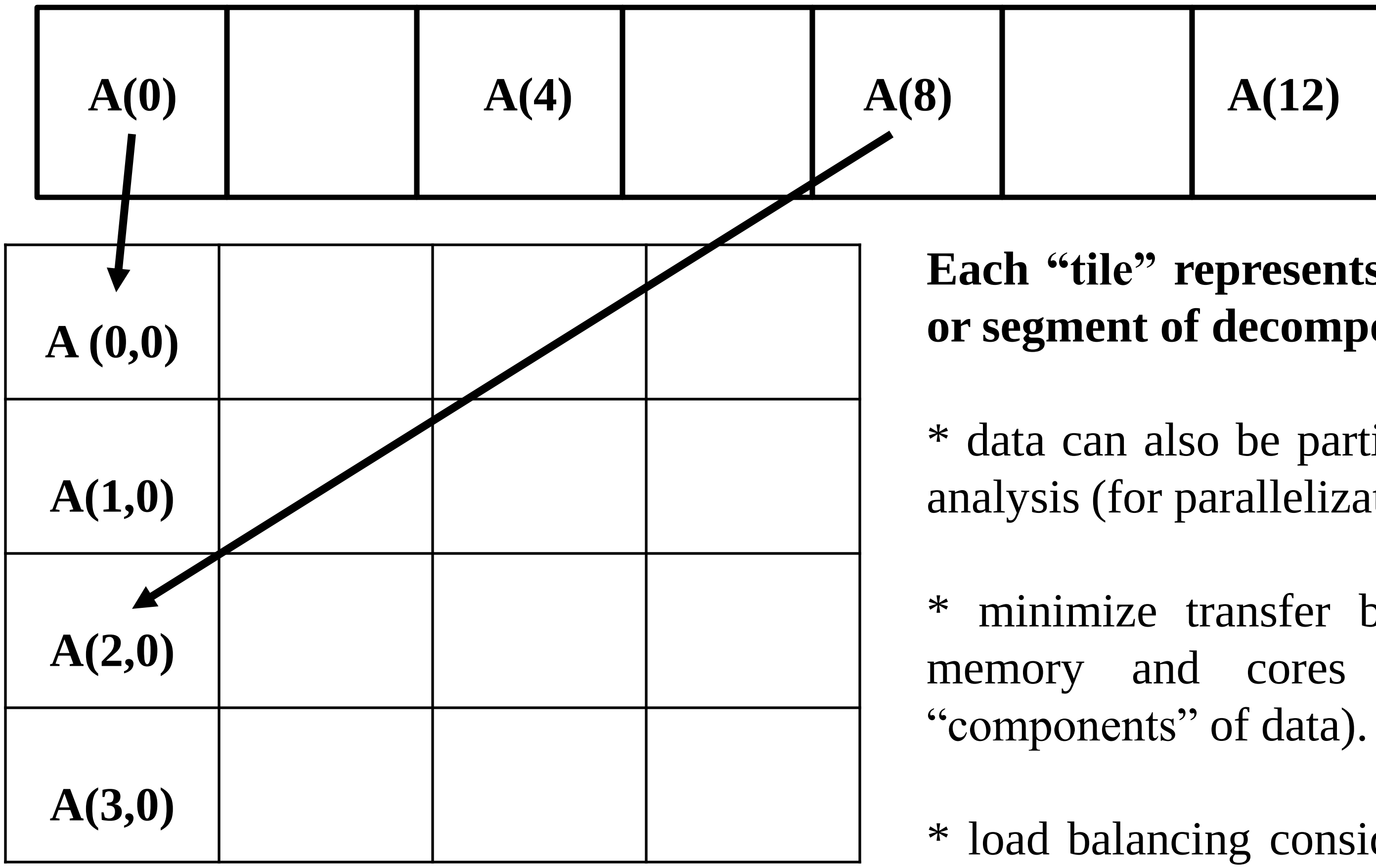
Multigrid techniques:

* grid resized as needed during convergence of optimization algorithm.

* appropriate for multiscalar computations (physics-based problems).

How can we statistically decompose a linear vector (a series of values for a single variable) in a parallel, graphical world? (e.g. PCA, Fourier transformations)

COURTESY: Hwu and Kirk, Programming Massively Parallel Processors: a hands on approach.



Original vector partitioned into tiles

Each "tile" represents a component or segment of decomposed data:

* data can also be partitioned prior to analysis (for parallelization).

* minimize transfer between global memory and cores (find natural "components" of data).

* load balancing considerations (each core should be given similar amount of work to do).

How do we implement useful algorithms in a graphical, parallel environment? Mergesort Algorithm for GPU

Order vector A[p,…r] into list (p → q → r).

Divide: 1) A [p,…q] and 2) A [q+1,….r]

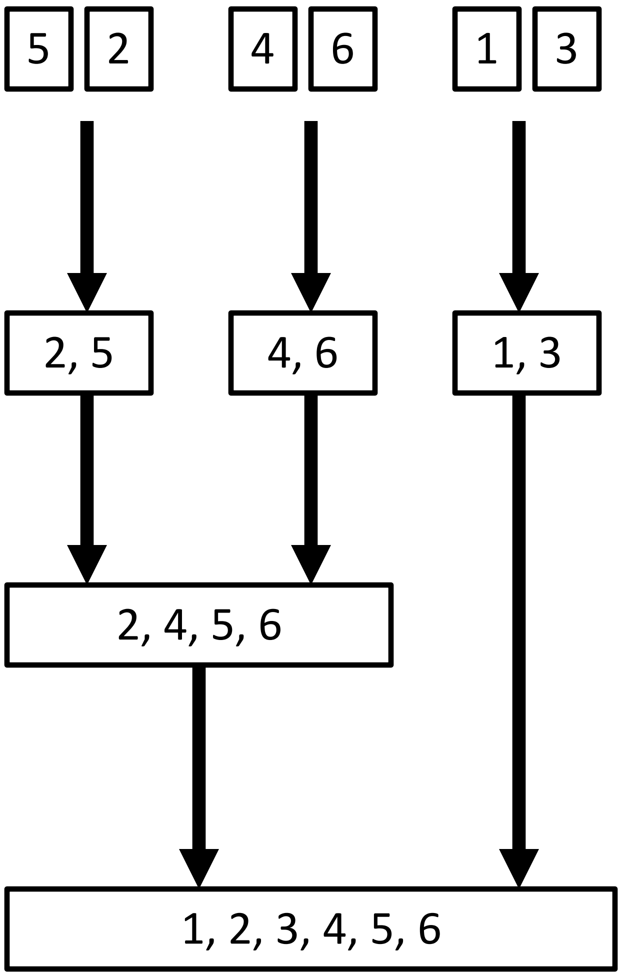
Conquer: sort 1) A [p,…q] and 2) A [q+1,….r]

Merge: 1) A [p,…q] and 2) A [q+1,….r] into A [p,r]

Not easy to implement true recursion in GPU, due to the lack of direct communication between cores.

* iteratively transfer increasingly longer strings to new processors, solve problem.

Iterative Version of Mergesort (Kukanas and Devine, CUDA Gems, 2009)



Biology ***in parallel***

What kind of biological problems can we address using a CUDA architecture?

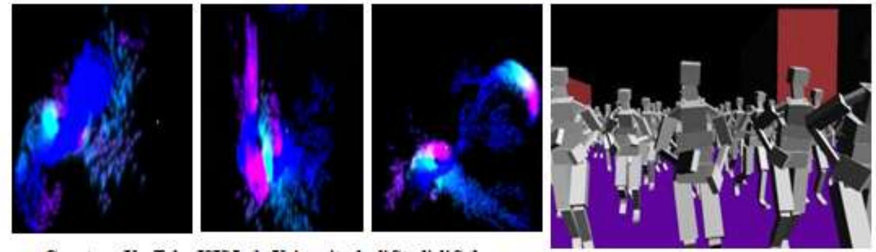
Distributed Behavior:
Flocking, n-body problems (interaction rules , agent-based approaches).

* well-suited problem to a parallel approach, but hard to implement.

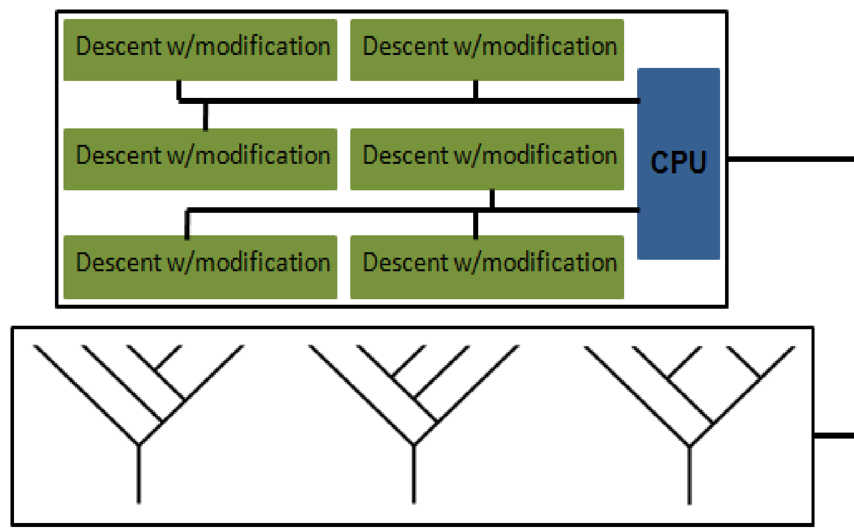
Phylogenetics:
Reconstruct the “true tree” of life – evolutionary relationships between species.

* parallel version of common algorithms to reconstruct, sort.

GPU-based Distributed Behavior Models with CUDA



Courtesy: YouTube, ISIS Lab, Universita degli Studi di Salerno



Phylogenetics on CUDA (Parallel) Architectures

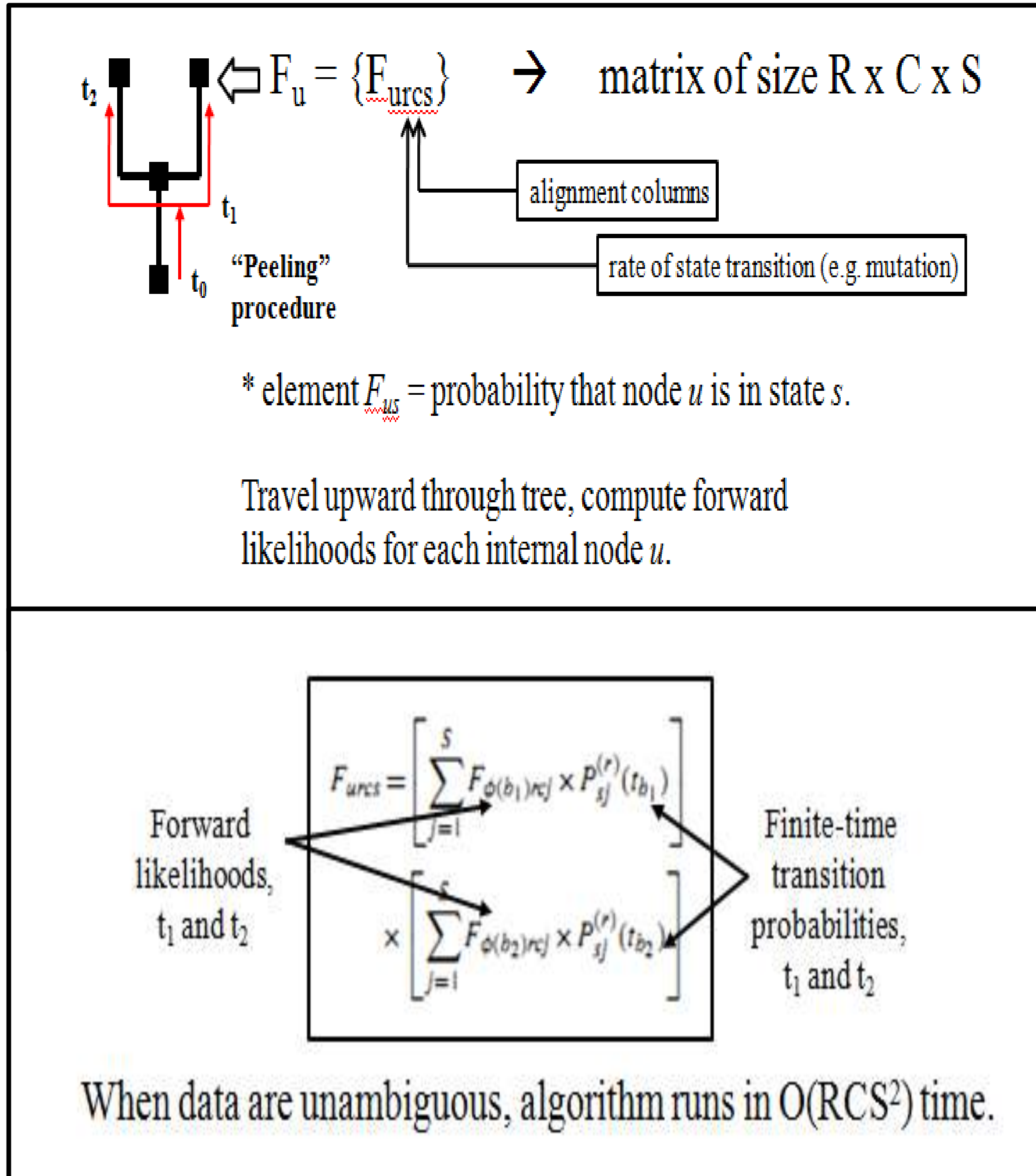
Inferrence of Phylogenetic Tree Topologies

Are parallel methods capable of reconstructing evolutionary relationships?

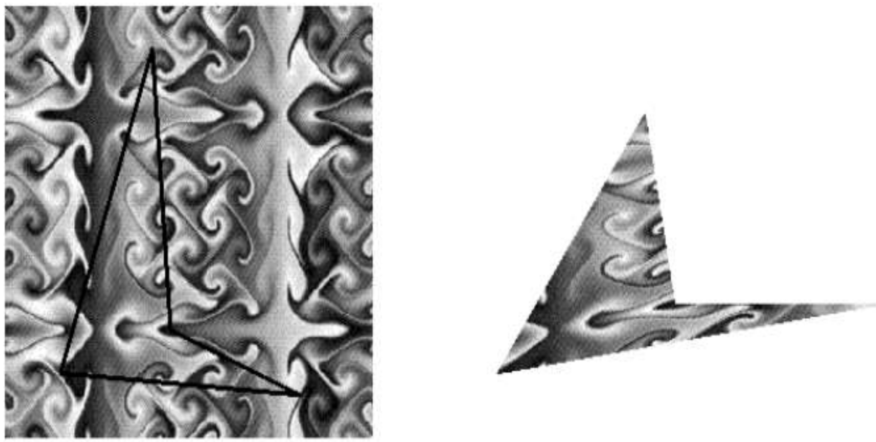
YES

Felsenstein's "Peeling" algorithm, as implemented by Suchard and Rambaut (Bioinformatics, 25(11), 1370-1376 – 2009).

Maximum likelihood is only marginally better on GPU than on parallel CPU.



Graphical biology in the form of texture maps



COURTESY: OpenGL Programming Guide, Chapter 9 (Texture Mapping)

A way to tile multidimensional surfaces with 2-D templates (in computer graphics context):

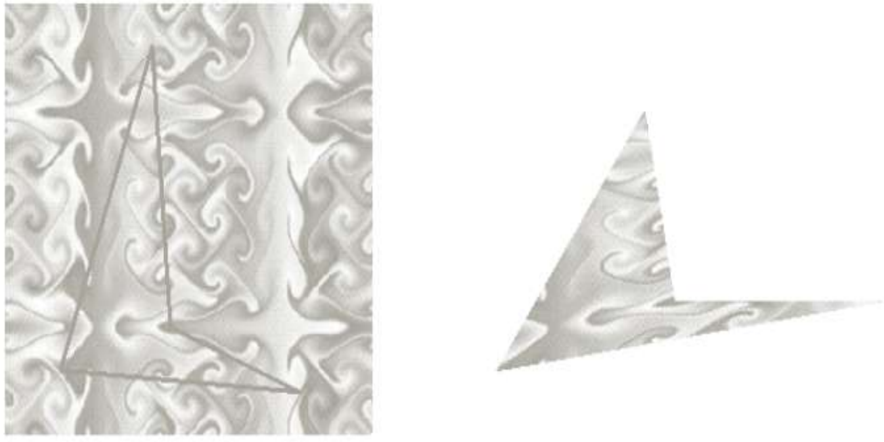
Adding detail to a surface:

* data mapped to a pattern.

* pattern is mapped to a surface.

* pattern consists of repeats, motifs.

Graphical biology in the form of texture maps



COURTESY: OpenGL Programming Guide, Chapter 9 (Texture Mapping)

A way to tile multidimensional surfaces with 2-D templates (in computer graphics context):

Adding detail to a surface:

* data mapped to a pattern.

* pattern is mapped to a surface.

* pattern consists of repeats, motifs.

What kind of problems can this and other data structures potentially address?

* **intra- and inter cellular signaling** (collective activities of signaling molecules).

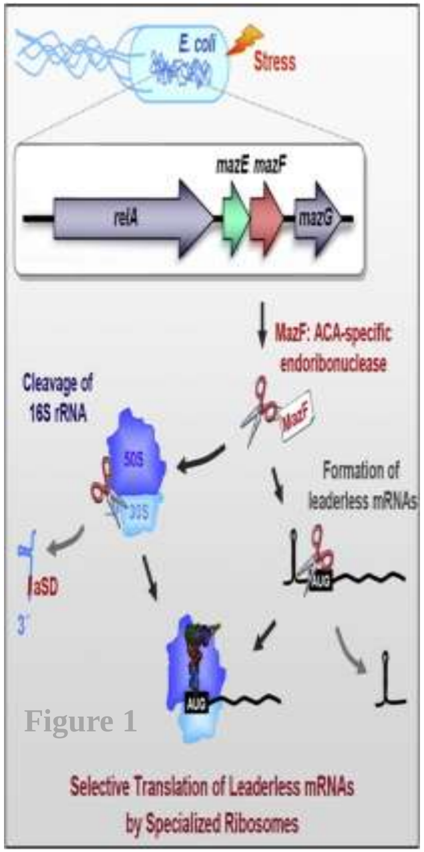
* **morphogenesis, gene expression, and proteomics** (where “sequence” and “form” has higher-dimensional information).

*** population-based problems** (populations of cells, organisms produce emergent structures, but behave autonomously).

* **multiscalar problems** (where processes occur at multiple scales).

Intra- and Intercellular Signaling

In leaderless mRNAs, 3' end is cleaved, modifies sites of action (Cell, 147(1), 147-157 – 2011).



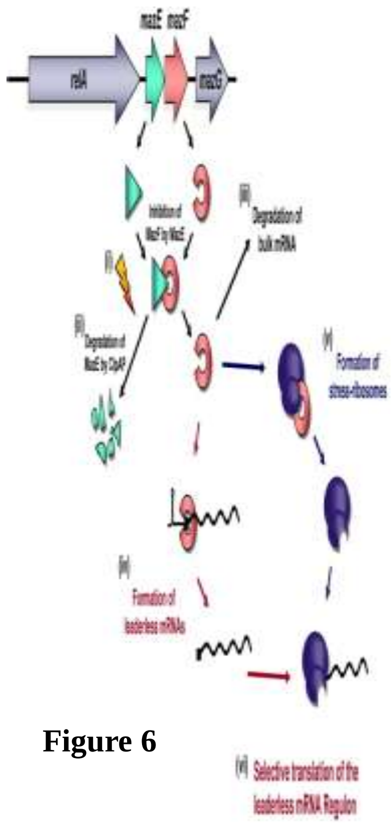


Figure 6

Interleukin 1-β – secretory protein lacking a signal peptide (special route to transport). Mol Bio. Cell, 10(5), 1463-1475 (1999).

Intra- and Intercellular Signaling

In leaderless mRNAs, 3’ end is cleaved, modifies sites of action (Cell, 147(1), 147-157 – 2011).

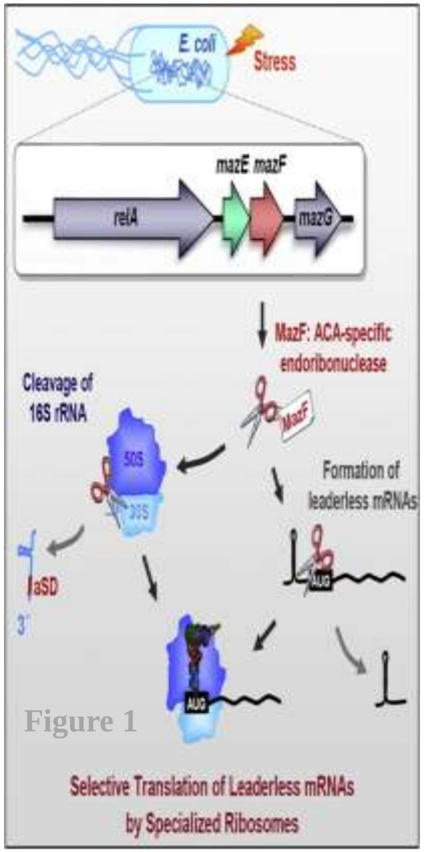


Figure 1

Selective Translation of Leaderless mRNAs by Specialized Ribosomes

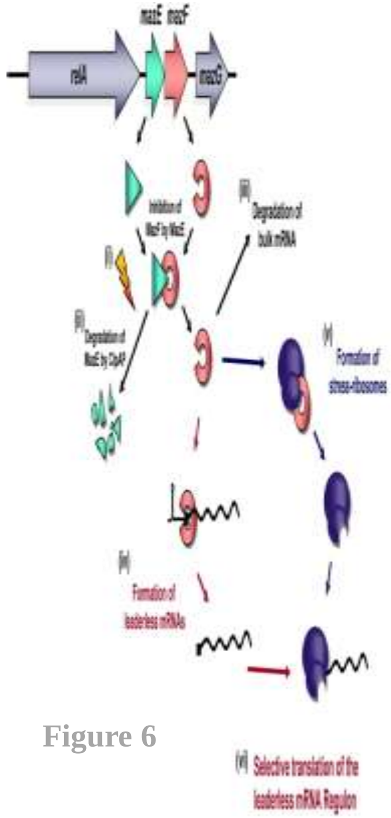
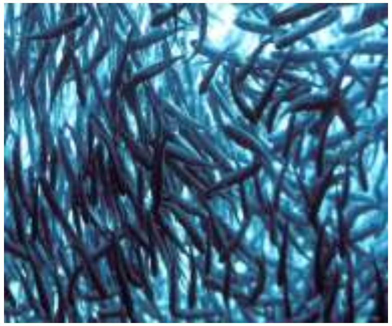


Figure 6

Interleukin 1-β – secretory protein lacking a signal peptide (special route to transport). Mol Bio. Cell, 10(5), 1463-1475 (1999).

What does it mean to be leaderless?

Occupy *x,y,z,t*?



Shoaling fish?

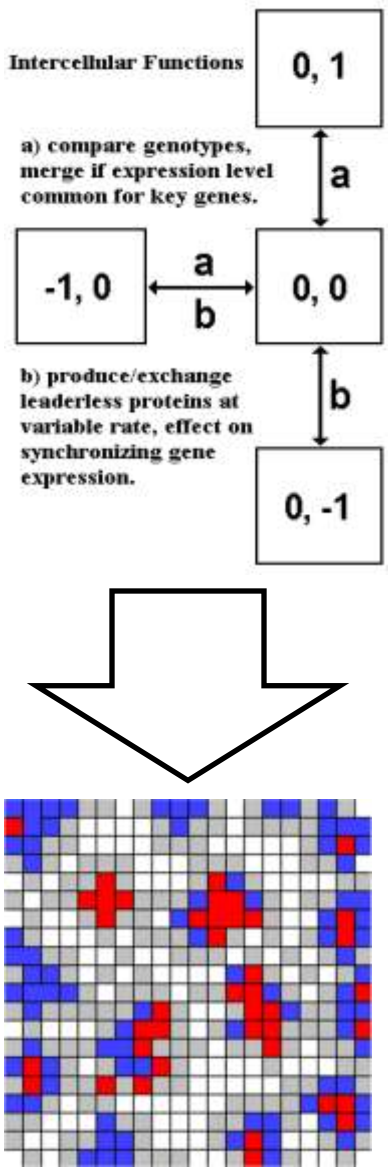
Leaderless: not embedded in a hierarchy.

Leaderless molecules do not follow the leader (but not necessarily random behavior).

* under certain conditions, leaderless activity (shoaling fish?) may lead to order, patterned behavior.

See also: **Intracellular signaling proteins as "smart" agents in parallel distributed processes**. BioSystems, 50, 159-171 (1999).

Example from Cellular Reprogramming (non GPU)



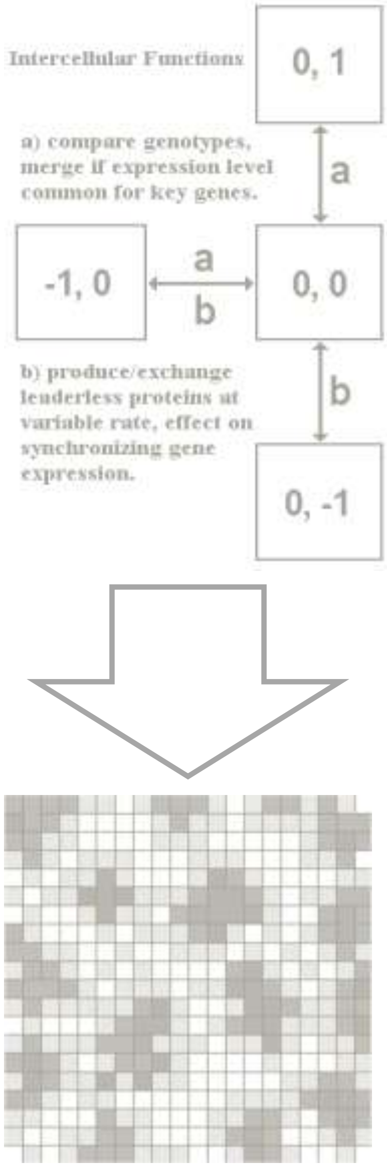
Dynamics Days 2012
Poster (left)

Complex Model:

* hybrid model simulates transforming cell population (lower left).

* interaction rules – series of intercellular functions (upper left).

Example from Cellular Reprogramming (non GPU)



Dynamics Days 2012 Poster (left)

Complex Model:

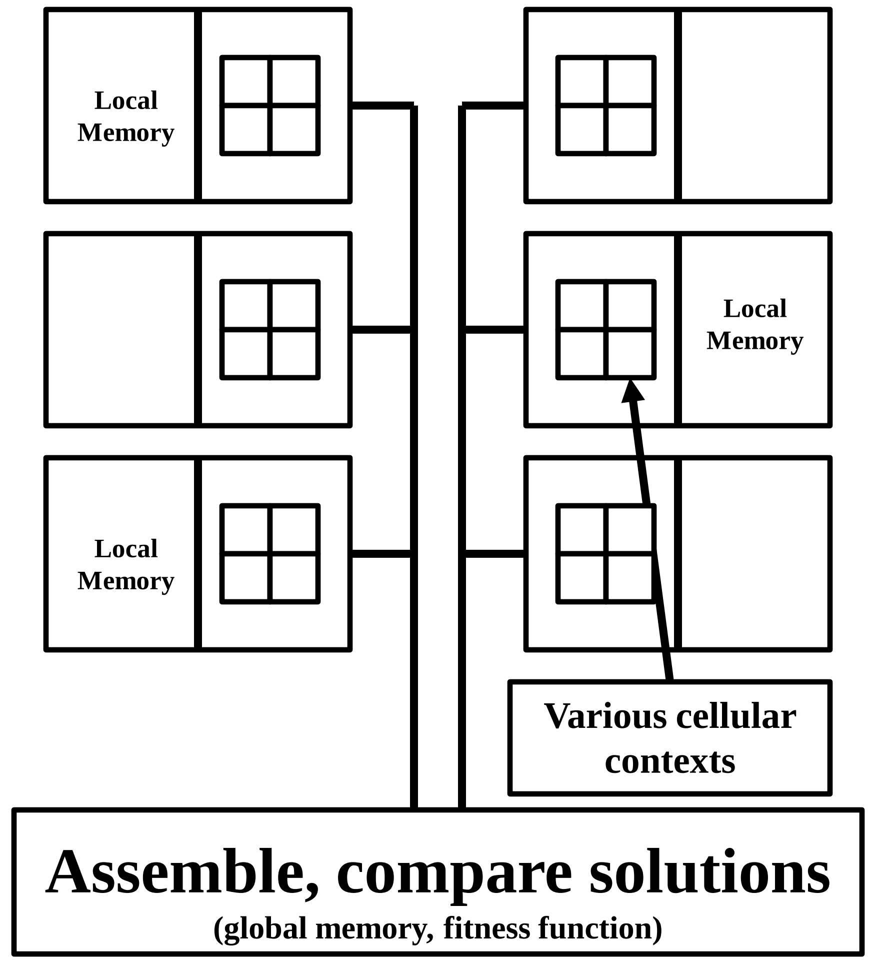
* hybrid model simulates transforming cell population (lower left).

* interaction rules – series of intercellular functions (upper left).

Map to a GPU:

***** simulate sets of CA neighborhoods with different "genetic" backgrounds.

Hybrid model (genetic algorithm, cellular automata): **GPU** may allow for cellular contexts to be tested, compared.



Gene Expression and Proteomics

Graphical, parallel computing can be used to analyze microarray data, epistatic interactions, feature detection in proteomics.

Hussong et.al (2009). Highly accelerated feature detection in proteomics data sets using modern graphics processing units. Bioinformatics, 25, 1937-1943.

Shterev et.al (2010). permGPU: Using graphics processing units in RNA microarray association. BMC Bioinformatics, 11, 329

Sinnott-Armstrong et.al (2009). Accelerating epistasis analysis in human genetics with consumer graphics hardware. BMC Bioinformatics, 2.

GOALS:

* find patterns (motifs)

* make predictions (alignments, predictive models).

* also to find functional relationships (epistasis).

* RNAseq (future): map structure (sequence) to function (expression).

Gene Expression and Proteomics

Graphical, parallel computing can be used to analyze microarray data, epistatic interactions, feature detection in proteomics.

Map biological process to a computational model to a GPU implementation.

Hussong et.al (2009). Highly accelerated feature detection in proteomics data sets using modern graphics processing units. Bioinformatics, 25, 1937-1943.

Shterev et.al (2010). permGPU: Using graphics processing units in RNA microarray association. BMC Bioinformatics, 11, 329

Sinnott-Armstrong et.al (2009). Accelerating epistasis analysis in human genetics with consumer graphics hardware. BMC Bioinformatics, 2.

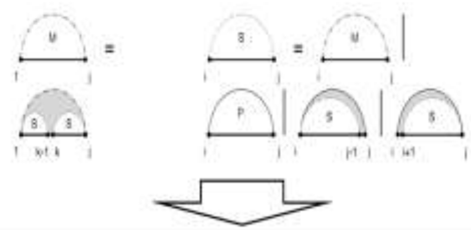
GOALS:

* find patterns (motifs)

* make predictions (alignments, predictive models).

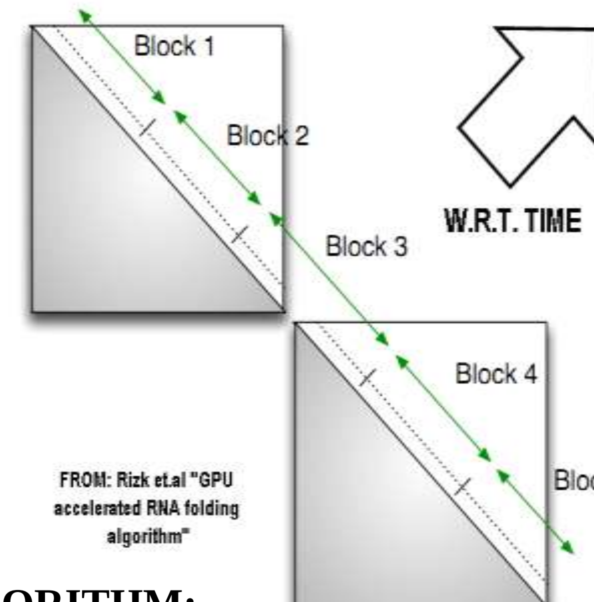
* also to find functional relationships (epistasis).

* RNAseq (future): map structure (sequence) to function (expression).



Algorithm 4 QM kernel, tiled implementation

```
BlockSize: Two dimension = BS · BS
Block Assignment: Computation of "middle part" of QM for a tile of BS · BS
  cells, with upper left corner (i0, j0)
tx ← threadIdx.x, ty ← threadIdx.y
iter ← beginning of "middle part"
bound ← end of "middle part"
temp ← ∞
while iter < bound do
  Load blocks in shared memory:
  Shared.L [tx][ty] ← QS(i0 + ty, iter + tx)
  Shared.C [tx][ty] ← QS(iter + 1 + ty, j0 + tx)
  Syncthreads
  for k = 0, k ≤ BS, k++ do
    temp ← min(temp,Shared.L [ty][k] +Shared.C [k][tx] )
  end for
  iter ← iter + BS
end while
QM_{i0+tx,j0+ty} ← temp
```



ALGORITHM:

Decompose, step through matrices of RNA folding data

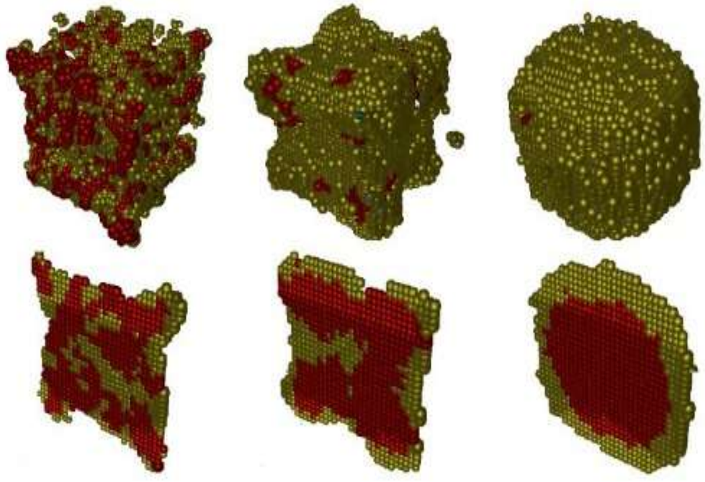
Parallel, Graphical Morphogenesis

Autonomous Cells (signaling pathways, genomes) >> assembly into communities, populations (emergent process).

Graphical approach allows us to compute many possible scenarios of growth, intercalation, and translation/rotation.

* development is explicitly 3D, physics engine capabilities of GPU well-suited to the task.

* modeling variability in development may also be possible (multicore architecture can allow you to introduce different conditions on different threads/cores).

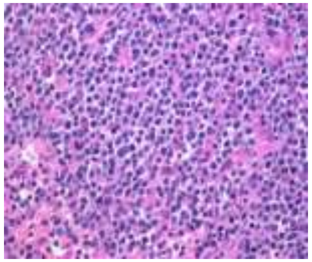


EXAMPLE:

Tapia, J.J. and D'Souza, R.M. (2011). Parallelizing the Cellular Potts Model on Graphics Processing Units. Computer Physics Communications, 182(4), 857-865.

* used CompuCell 3D to model morphogenesis using Cellular Potts model.

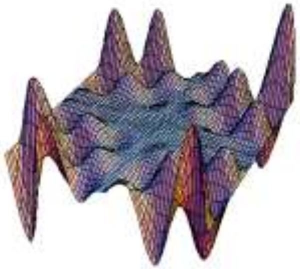
Population Modeling in Parallel

Populations:
* ubiquitous in biology, particularly for adaptive processes.

* agents distributed across nodes, checked against solutions (in global memory).

Genetic Algorithms (GA):
* uses a population of agents to find the local, global optima for a given problem.



* can be applied to optimization-friendly problems.

Population Modeling in Parallel



Populations:

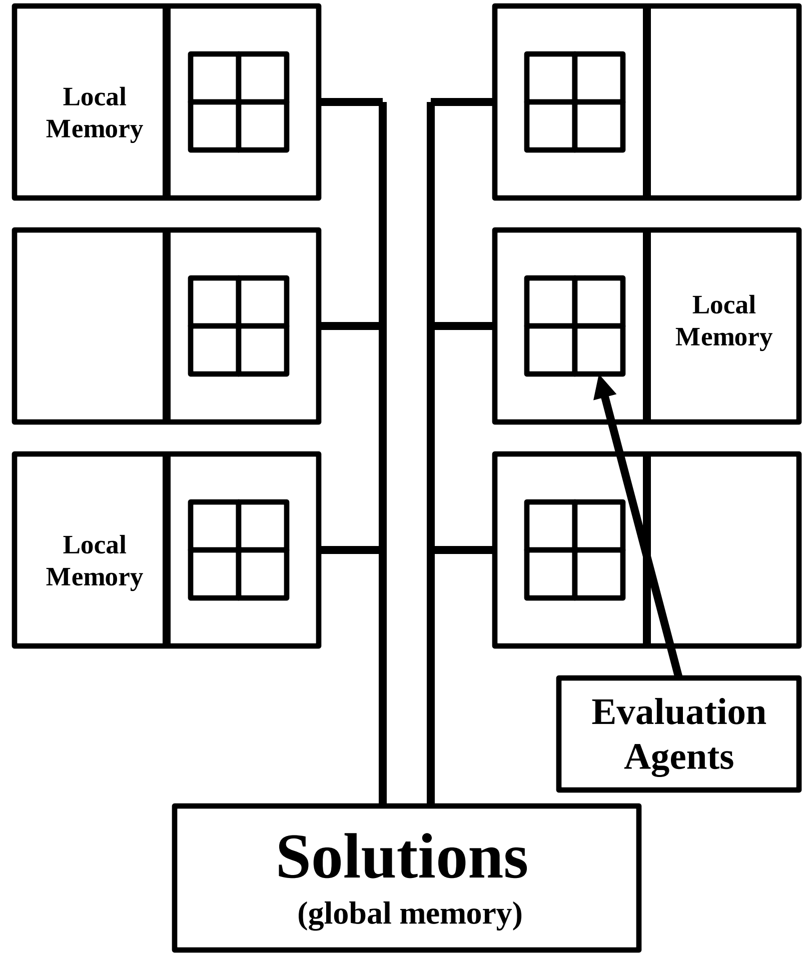
* ubiquitous in biology, particularly for adaptive processes.

* agents distributed across nodes, checked against solutions (in global memory).

Genetic Algorithms (GA):

* uses a population of agents to find the local, global optima for a given problem.

* can be applied to optimization-friendly problems.



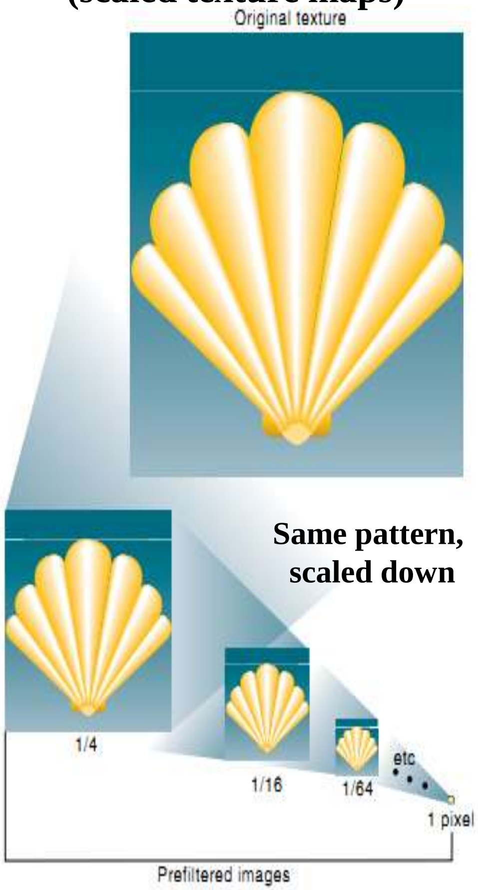
COURTESY: Chapter 6, Parallel Combinatorial Optimization

Populations can be averaging, filtering devices for stochasticity at single cell, organism level.

Multiscalar Processes

Using Mipmaps

(scaled texture maps)



COURTESY: OpenGL Programming Guide, Chapter 9 (Texture Mapping)

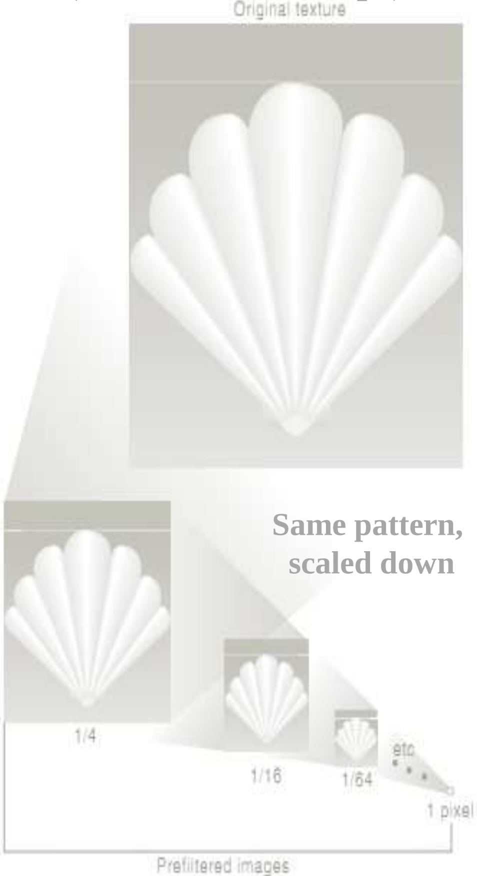
In some cases, multigrid methods are used for multiscalar problems.

BUT CONSIDER:

Processes that occur at different scales, unified by a scaling factor?

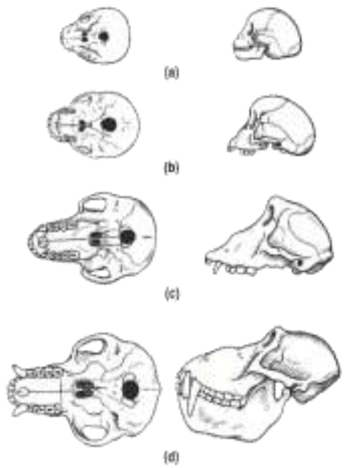
Multiscalar Processes

Using Mipmaps
(scaled texture maps)



Same pattern, scaled down

COURTESY: OpenGL Programming Guide, Chapter 9 (Texture Mapping)



Scaled growth, Babboon skulls
[Basics of Allomtery, Pharyngula](#) Blog

Fractal growth

In some cases, multigrid methods are used for multiscalar problems.

BUT CONSIDER:

Processes that occur at different scales, unified by a scaling factor?

Variation at the same scale (allometric growth):

* growth can be scaled by a exponential factor (2/3rds scaling, etc).

* same pattern of genes expression, just goes on longer (growth):

Self-similarity at different scales (fractal growth):

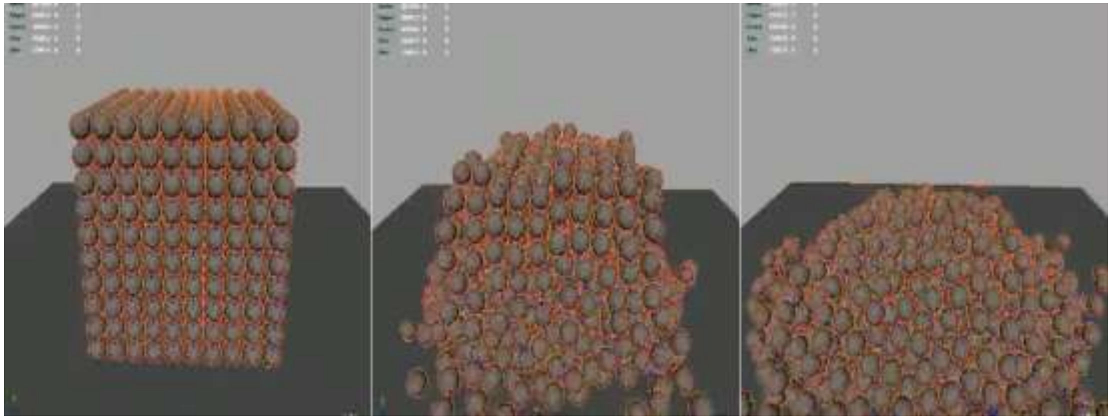
* features are similar at different scales, just larger or smaller.

In a parallel universe?

Graphics or no graphics, is that the question (benefit)?

What are the benefits of graphical, parallel processing (besides speed-up)?

* can handle matrix-intensive calculations (co-registration, virtual physics, volume rendering) well.



PhysX MAYA demo. COURTESY: YouTube.

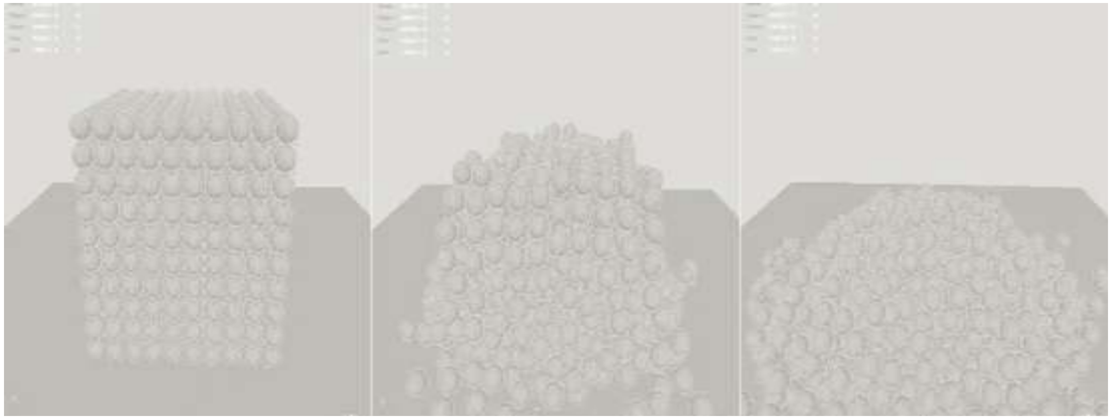
Graphics or no graphics, is that the question (benefit)?

What are the benefits of graphical, parallel processing (besides speed-up)?

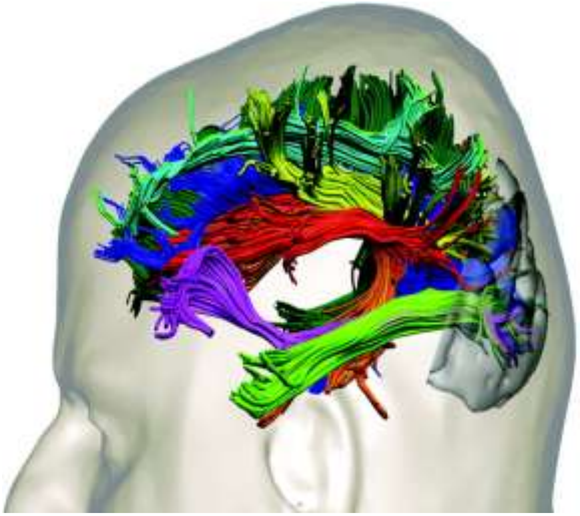
* can handle matrix-intensive calculations (co-registration, virtual physics, volume rendering) well.

* can handle multi-dimensional datasets (complex geometries) well.

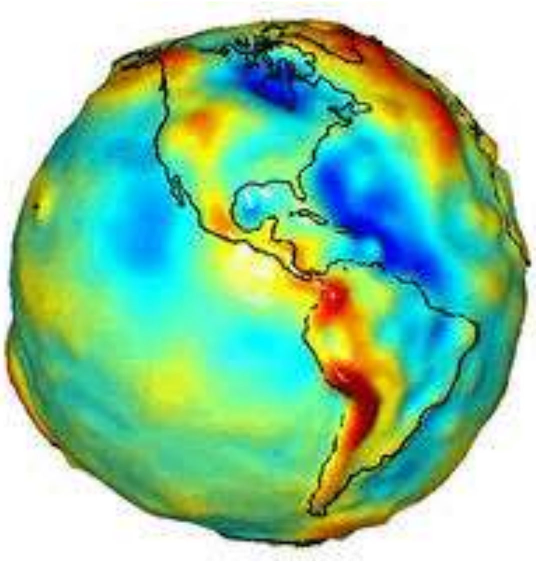
* parallelization = new set of rules (requires new algorithms well suited to distributed processes).



PhysX MAYA demo. COURTESY: YouTube.



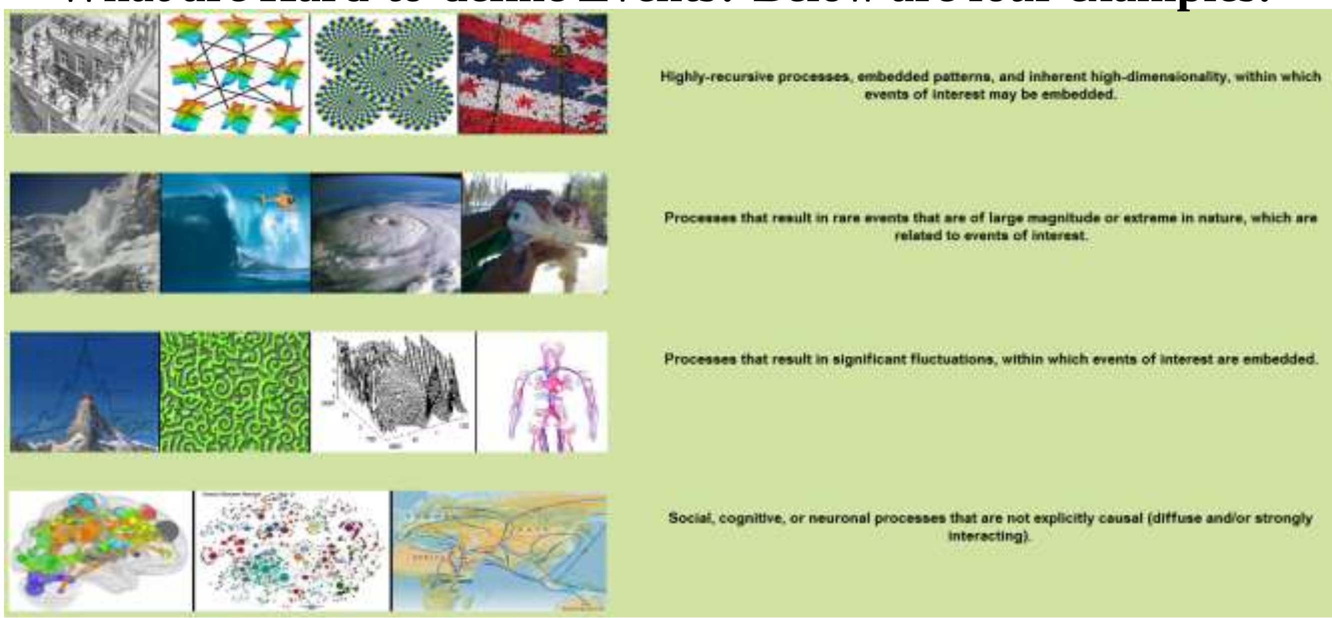
Co-registered datasets: neuroimaging data (above), magnetic field mapping (below).



Hard-to-define-Events Workshop Announcement (July, 2012)

Are you interested in how complexity obscures the important features of a complex system? Do you work with a system highly affected by or embedded in noise? Do you work with a system that is fundamentally interactive? Has it impeded your science? Such hard-to-define events are not a major focus of current scientific approaches. With this workshop, we will begin to challenge conventional wisdom of statistical analysis and modeling by considering a variety of models (robots, organisms, cells) and techniques (parallel computing, nonlinear dynamics). We are seeking participants from multiple perspectives, and examples that transcend traditional disciplines are of particular interest.

What are Hard-to-define Events? Below are four examples:



Held in conjunction with the Artificial Life 13 conference, hosted by the BEACON Center at Michigan State University.

WANTED! Virtual participants (anywhere in the world) on the weekend of July 20-22, 2012.

Contact Bradly Alicea (biodroid) at bradly.alicea <at> ieee.org for more information, or visit http://www.msu.edu/~aliceabr/hard-to-define-events.htm for the most current developments and scheduling updates.