Computing Beyond Silicon Summer School

Physics becomes the computer

Norm Margolus
Physics becomes the computer

Emulating Physics

» Finite-state, locality, invertibility, and conservation laws

Physical Worlds

» Incorporating comp-universality at small and large scales

Spatial Computers

» Architectures and algorithms for large-scale spatial computations

Nature as Computer

» Physical concepts enter CS and computer concepts enter Physics
Review: CA’s with conservations

To make reversibility and other conservations manifest, we employ a multi-step update, in each step of which either

1. The data are rearranged without any interaction, or
2. The data are partitioned into disjoint groups of bits that change as a unit. Data that affect more than one such group don’t change.

We would like to be able to study large-scale computations with this kind of conservation-friendly format, which allows them to map efficiently onto microscopic physics, and also allow them to have interesting macroscopic behavior.

CBSSS 6/26/02
Spatial Computers
Spatial Computers

- Finite-state computations on lattices are interesting
- Crystalline computers will one day have highest performance
- Today’s ordinary computers are terrible at these computations

*We may one day compute with regular crystals of atoms.*
CAMs 1 to 6

- Tom Toffoli and I wanted to see CA-TV
- Stimulated by new rev rules
- Rule = a few TTL chips!
- Later, rule = SRAM LUT
- LGA’s needed more “serious” machine!

CBS SS 6/26/02
CAM-8 Programming Model

Data movement step:

Site update step:

CBSSS 6/26/02
CAM-8 node

- maps directly onto programming model (bit fields)
- 16-bit lattice sites (use internal dimension to get more)
- each node handles a chunk of a larger space

CBSSS 6/26/02
Lattice example

/* Four direction lattice gas */

2D-square-lattice {512 512}
create-bit-planes {N S E W}
define-step
    shift-planes {{(N,0,-1) (S,0,1)}
                   {(E,1,0) (W,-1,0)}}
    for-each-site
        if {N==S && E==W}
            {xchng(N,E); xchng(S,W)}
        endif
    end-site
end-step

CBSSS 6/26/02
Lattice example

/* Four direction lattice gas */

2D-square-lattice {512 512}
create-bit-planes {N S E W}
define-step
    shift-planes {(N,0,-1) (S,0,1) (E,1,0) (W,-1,0)}
    for-each-site
        if {N==S && E==W}
            {xchng(N,E); xchng(S,W)}
        endif
    end-site
end-step

CBS 5/26/02
Lattice example

/* Four direction lattice gas */

2D-square-lattice {512 512}
create-bit-planes {N S E W}
define-step
    shift-planes { (N,0,-1) (S,0,1) (E,1,0) (W,-1,0) }
    for-each-site
        if { N==S && E==W }
            xchng(N,E); xchng(S,W)
        endif
    end-site
end-step

CBSSS 6/26/02
Lattice example

/* Four direction lattice gas */

2D-square-lattice {512 512}
create-bit-planes {N S E W}
define-step
    shift-planes {((N,0,-1) (S,0,1))
                   (E,1,0) (W,-1,0)}
    for-each-site
        if {N==S &amp; E==W}
           {xchng(N,E); xchng(S,W)}
        endif
    end-site
end-step
Lattice example

/* Four direction lattice gas */

2D-square-lattice {512 512}
create-bit-planes {N S E W}
define-step
    shift-planes { (N,0,-1) (S,0,1) (E,1,0) (W,-1,0) }
    for-each-site
        if { N==S & E==W }
            xchng(N,E); xchng(S,W)
        endif
    end-site
end-step
Lattice example

/* Four direction lattice gas */
2D-square-lattice {512 512}
create-bit-planes {N S E W}
define-step
  shift-planes {(N,0,-1) (S,0,1)
               (E,1,0) (W,-1,0)}
for-each-site
  if {N==S && E==W}
    {xchng(N,E); xchng(S,W)}
  endif
end-site
end-step
Lattice example

/* Four direction lattice gas */

2D-square-lattice {512 512}
create-bit-planes {N S E W}
define-step
    shift-planes {((N,0,-1) (S,0,1))
                   (E,1,0) (W,-1,0)}
    for-each-site
        if {N==S && E==W}
            xchng(N,E); xchng(S,W)
        endif
    end-site
end-step

CBSSS 6/26/02
CAM-8: discrete hydrodynamics

Six direction LGA flow past a half-cylinder. System is 2Kx1K.

CBSSS 6/26/02
CAM-8: EM scattering

\[ E_z \text{ for ILGA (x10^{-3})} \]

\[ E_z \text{ for TLM (x10^{-3})} \]

\[ 0 \quad 50 \quad 100 \quad 150 \quad 200 \quad 250 \quad 300 \quad 350 \quad 400 \]

\[ \text{time (\Delta t)} \]

TLM (200x200 mesh)
ILGA (400x400 mesh)
CAM-8: materials simulation

512x512x64 spin system, 6up/sec w rendering

256x256x256 reaction-diffusion (Kapral)

CBSSS 6/26/02
CAM-8: logic simulation

- Physical sim
- Logical sim
- Wavefront sim
- Microprocessor at right runs at 1KHz

A simple logic simulation in a gate-array-like lattice dynamics.

Wavefront simulation (R. Agin)
CAM-8: porous flow

- Used MRI data from a rock (Fontainebleau sandstone)
- Simulation of flow of oil through wet sandstone agrees with experiment within 10%
- 3D LGA’s used for chemical reactions, other complex fluids
CAM-8: image processing

- Local dynamics can be used to find and mark features
- Bit maps can be “continuously” rotated using 3 shears

Three-dimensional rotation (Euler angles: $\alpha = -10^\circ$, $\beta = 30^\circ$, $\gamma = -15^\circ$) achieved by three shears.
CAM-8 node (1988 technology)

- 25 Million 16-bit site-updates/sec
- 8 node prototype
- arbitrary shifts of all bit fields at no extra cost
- limited by mem bandwidth
- still 100x PC!

6MB/sec x 16 DRAMs + 16 ASICs

25 MHz
64Kx16

CBSSS 6/26/02
FPGA node design (1997)

- 100 x faster per DRAM chip
- No custom hardware
- Needed FPGA interface to RDRAM

- 600MB/sec
- 1 RDRAM + 1 FPGA

150 MHz
64Kx16 pipelined
Embedded DRAM (2000)

4 Mbit Block of DRAM:

2K

2K

2K bits / 40 ns

20 Blocks of DRAM (80 Mbits total):

40K

2K

40K bits / 40 ns = 1 Tbit/sec per chip!

Each chip would be 1000x faster than 128-DRAM CAM-8!
SPACERAM: A general purpose crystalline lattice computer

- Large scale prob with spatial regularity
- 1 Tbit/sec $\approx 10^{10}$ 32-bit mult-accum / sec (sustained) for finite difference
- Many image processing and rendering algorithms have spatial regularity.
- Brute-force physical simulations (complex MD, fields, etc.)
- Bit-mapped 3D games and virtual reality (simpler and more direct)
- Logic simulation (physical, wavefront)
- Pyramid, multigrid and multiresolution computations

\[ f_{x,y,t+1} = \alpha f_{x+1,y,t} + \beta f_{x-1,y,t} + \gamma f_{x,y+1,t} \\
+ \delta f_{x,y-1,t} + \epsilon f_{x+1,y+1,t} + \zeta f_{x+1,y-1,t} \\
+ \eta f_{x-1,y+1,t} + \iota f_{x-1,y-1,t} + \kappa f_{x+2,y,t} \\
+ \mu f_{x-2,y,t} + \nu f_{x,y+2,t} + \sigma f_{x,y-2,t} \\
+ \tau f_{x,y,t} \]
SPACERAM

- **The Task:** Large-scale brute force computations with a crystalline-lattice structure
- **The Opportunity:** Exploit row-at-a-time access in DRAM to achieve speed and size.
- **The Challenge:** Dealing efficiently with memory granularity, commun, proc
- **The Trick:** Data movement by layout and addressing

4Mbits/DRAM, 256 bit I/O ×20 @200 MHz → 1 Tbit/sec, 10 MB, 130 mm² (.18μm) and 8 W (mem)

CBSSS 6/26/02
Mapping a lattice into hardware

- Divide the lattice up evenly among mesh array of chips
- Each chip handles an equal sized sector of the emulated lattice
Mapping a lattice into hardware
Mapping a lattice into hardware

- Use blocks of DRAM to hold the lattice data (Tbits/sec)
- Use virtual processors for large comps (depth-first, wavefront, skew)
- **Main challenge**: mapping lattice computations onto granular memory
Lattice computation model

1D example: the rows are bit-fields and columns sites

Shift bit-fields periodically and uniformly (may be large)

Process sites independently and identically (SIMD)
Mapping the lattice into memory
Mapping the lattice into memory

CAM-8 approach: group adjacent into memory words

Shifts can be long but re-aligning messy → chaining

Can process word-wide (CAM-8 did site-at-a-time)

CBSSS 6/26/02
Mapping the lattice into memory

\[
\begin{align*}
\text{[ASCII Art]} & \quad = & \quad \text{[ASCII Art]} \\
+ & \quad \text{[ASCII Art]} & \quad \text{[ASCII Art]} \\
+ & \quad \text{[ASCII Art]} & \quad \text{[ASCII Art]} \\
+ & \quad \text{[ASCII Art]} & \quad \text{[ASCII Art]}
\end{align*}
\]

**CAM-8:** Adjacent bits of bit-field are stored together
Mapping the lattice into memory

\[ \begin{array}{c}
\quad \\
\quad \quad = \\
\quad + \\
\quad + \\
\quad + \\
\end{array} \]

\emph{SPACERAM:} bits stored as evenly spread \emph{skip} samples.
Processing skip-samples

- Process sets of evenly spaced lattice sites (*site groups*)
- Data movement is done by reading shifted data
Processing skip-samples

A[0]
B[0]
A[1]
B[1]
A[2]
B[2]
A[3]
B[3]

CBSSS 6/26/02
Processing skip-samples

A[0]
B[0]
A[1]
B[1]
A[2]
B[2]
A[3]
B[3]

CBSSS 6/26/02
Processing skip-samples

CBSSS 6/26/02
Processing skip-samples

A[0]  A[0]
B[0]  B[0]


CBSSS 6/26/02
Processing skip-samples

CBSSS 6/26/02
Processing skip-samples

A[0]
B[0]

A[1]
B[1]

A[2]
B[2]

A[3]
B[3]

A[2]
B[0]
Processing skip-samples

\[ A[0] \]
\[ B[0] \]

\[ A[1] \]
\[ B[1] \]

\[ A[2] \]
\[ B[2] \]

\[ A[3] \]
\[ B[3] \]

CBSSS 6/26/02
Processing skip-samples
Processing skip-samples

A[0]  B[0]

Processing skip-samples
Processing skip-samples


CBSLS 6/26/02
DRAM module

- Assume hardware word size is 64-bits
- Assume we can address any word
- For each site group: address next word we need; rot if necessary
- Each bit field resides in a different module
Problem: granularity

- DRAM block has structure: e.g., all words in a row must be used before switching rows

Solution: grouping

\[
\text{-------------------} = ( \text{-------------------} + \text{-------------------} ) \\
+ ( \text{-------------------} + \text{-------------------} )
\]
DRAM module

- 3 levels of granularity: 2K-bit row, 256-bit word, 64-bit word
- All handled by data layout, addressing, and rot within 64-word
Gluing chips together

- All chips operate in lockstep
- Shift is periodic within each sector
- To glue sectors, bits that stick out replace corresponding bits in the next sector
DRAM module

- Any data that wraps around is transmitted over the mesh
- Corresponding bit on corresponding wire is substituted for it
- This is done sequentially in each of the three directions

row addr → DRAM block
  2K×8×256

col addr →

sub-addr → 64x4:64

shift amt → Barrel Rotator

wrap info → Mesh xyz Subst

64

64

to processor
Partitioning a 2D sector
Partitioning a 2D sector
The SPACERAM chip

- .18 μm CMOS DRAM proc. (20W, 215 mm²)
- 10% of memory bandwidth → control
- Memory hierarchy (external memory)
- Single DRAM address space
The SPACERAM chip
The SPACERAM chip
SPACERAM: symbolic PE

- Computation by LUT
- Permuter lets any DRAM wire play any role
- MUX lets LUT output be used conditionally
- LUT is really a MUX, with data bussed
SPACERAM: symbolic PE
Conclusions

- Addressing is powerful
- Simple hardware provides a general mechanism for lattice data movement
- We can exploit the parallelism of DRAM access without extra overhead

18μm CMOS, 10 Mbytes DRAM, 1 Tbit/sec to mem, 215 mm², 20W

http://www.ai.mit.edu/people/nhm/isca.pdf
Physics becomes the computer

*Emulating Physics*
- Finite-state, locality, invertibility, and conservation laws

*Physical Worlds*
- Incorporating comp-universality at small and large scales

*Spatial Computers*
- Architectures and algorithms for large-scale spatial computations

*Nature as Computer*
- Physical concepts enter CS and computer concepts enter Physics

CBS 6/26/02