Programmable Cellular Automata

End-User Parallel Programming (EUPP) to Map Naturally Parallel Applications → Massively Parallel Computer Hardware Architectures

Dr. John-Thones Amenyo
jta@york.cuny.edu
Dept. Math & CS
July 2009

# Programmable Cellular Automata

## Intermediate Formalism

## Map: Application Trends $\rightarrow$ Architecture Trends

<table>
<thead>
<tr>
<th>Application</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed Sensor Networks</td>
<td>Parallel Computers</td>
</tr>
<tr>
<td>Distributed Robot Systems</td>
<td>Distributed Computers</td>
</tr>
<tr>
<td>Natural Computation</td>
<td>Multi-core Computers</td>
</tr>
<tr>
<td>Autonomic Computing</td>
<td>Many-core Computers</td>
</tr>
<tr>
<td>Parallel Computing</td>
<td>&quot;Bronto” Scale Computing</td>
</tr>
</tbody>
</table>

**"Bronto” Scale Computing Spectrum & Gradations:**

- Exa-Scale
- Peta-Scale
- Tera-Scale
- Giga-Scale Computing

## Programmable Cellular Automata Architecture

<table>
<thead>
<tr>
<th></th>
<th>P: Processing</th>
<th>M: Memory</th>
<th>S: Switching</th>
<th>C: Control Coord.</th>
<th>IO: Input / Output Pixels, Acels etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>K (kilo)</td>
<td>KIPS</td>
<td>KBytes</td>
<td>Kbps</td>
<td></td>
<td>Kpixels</td>
</tr>
<tr>
<td>M (Mega)</td>
<td>MFLOPS</td>
<td>MBytes</td>
<td>Mbps</td>
<td></td>
<td>Mpixels</td>
</tr>
<tr>
<td>G (Giga)</td>
<td>GFLOPS</td>
<td>GBytes</td>
<td>Gbps</td>
<td></td>
<td>Gpixels</td>
</tr>
<tr>
<td>T (Tera)</td>
<td>TFLOPS</td>
<td>TBytes</td>
<td>Tbps</td>
<td></td>
<td>Tpixels</td>
</tr>
<tr>
<td>P (Peta)</td>
<td>PFLOPS</td>
<td>PBytes</td>
<td>Pbps</td>
<td></td>
<td>Ppixels</td>
</tr>
<tr>
<td>E (Exa)</td>
<td>EFLOPS</td>
<td>EBytes</td>
<td>Ebps</td>
<td></td>
<td>Epixels</td>
</tr>
</tbody>
</table>

IPS – Instructions/Sec; Bps – Bits/Sec; Pixels – Picture elements; Acels – Action elements, FLOPS – Floating Point Operations/Sec; Cops – Control Operations/Sec: Via Parallel / Modular / Distributed Designs
Programmable Cellular Automata Applications

<table>
<thead>
<tr>
<th>Computational STEM; Grand Challenges; Simulations; Sci. Data Visualization (Viz.); Bioinformatics; Neuroinformatics; Computational Connectomics; Immuno-informatics; Medical Informatics; VLDB; Large-scale RFID; GIS; Digital Data, Content, Media, Multimedia; Large scale Modeling; Large scale Data Mining</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPC; Distributed Sensor Networks; Distributed Robot Networks; Sensate Media; Multi-Touch Displays; Computational Intelligence</td>
</tr>
<tr>
<td>Network Computing; Internet; Web Services; Cloud Computing; Network Storage; Grid Computing; Tera-Grid; MPI; CC++; Ct</td>
</tr>
<tr>
<td>Desktop Tera-scale computing; Antenna Arrays; SDR; Distributed Sensor / Robot Networks; Ad hoc networks; VLSI Algorithmics, FPGA, DSP, ASIC chips, Reconfigurable Computers</td>
</tr>
</tbody>
</table>


## Programmable Cellular Automata Applications

<table>
<thead>
<tr>
<th>Computational Biology</th>
<th>Sequence Matching, Comparison, Analysis; (Virtual) Models &amp; Simulations: Cell, Biochemical / Metabolic Pathways, Gene Expression Networks, Immune Systems, Brain; Protein Folding / Conformation / Dynamics; <em>In Silico</em> Experiments.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computational Chemistry</td>
<td>Drug Design; Pharmaco-genomics; Modeling: Protein Functions; Protein Engineering</td>
</tr>
<tr>
<td>Physics &amp; Astronomy</td>
<td>Search for Exo-planets; Modeling: Stellar &amp; Galactic Evolution and Dynamics; Hydrodynamics, Fluid dynamics; Molecular Dynamics</td>
</tr>
<tr>
<td>Earth Science; Oceanography</td>
<td>Signal Processing: Remote Sensing, Radar, Lidar, Sonar; Modeling: Climate Change; Weather dynamics, prediction/forecasting</td>
</tr>
<tr>
<td>Env. / Ecol. Science</td>
<td>Modeling, Simulations, Signal Processing</td>
</tr>
<tr>
<td>Economics</td>
<td>Modeling: Econometrics; Financial; Global / Regional / National Econ.</td>
</tr>
<tr>
<td>Engineering/ ICT/IT</td>
<td>“@Home” Projects, PC Harvesting; Data Mining; Distributed Robotics; Cryptography; Nanomaterials modeling, Desktop supercomputing.</td>
</tr>
<tr>
<td>Military</td>
<td>Distributed Sensor Clouds; Battlefield Management Systems; Logistics</td>
</tr>
<tr>
<td>Consumer</td>
<td>Games; Image / Multi-media / Digital Content Processing; Lifetime Data Proc., Entertainment</td>
</tr>
</tbody>
</table>

## Programmable Cellular Automata Applications

**IT Consumers:** Home & Institutions: Desktop Supercomputing, Image, Photo, Picture, Video, Media, Multimedia Processing, Biometrics Processing

**Factories, Stores:** Distributed Sensors; RFID Data Collection / Storage

**Ships, Carriers, Tankers:** Signal Processing, Image Processing

**Aircraft:** Signal Processing, Image Processing

**Spacecraft:** Signal Processing

**Battlefields:**

**Env., Fields, Plots of Land, Airspace, Waterbodies:** Env. Monitoring; Flora & Fauna tracking; Pollution management

**Sea, Ocean Segments:**
Programmable Cellular Automata

Map: Application $\rightarrow$ Architecture
What Maps Into What? Why Does It Matter?

<table>
<thead>
<tr>
<th>Application</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Processors</td>
</tr>
<tr>
<td>Data Structures</td>
<td>Co-Processors</td>
</tr>
<tr>
<td>Functions</td>
<td>Processor Clusters</td>
</tr>
<tr>
<td>Operations</td>
<td>Memory, Storage</td>
</tr>
<tr>
<td>Routines, Scripts, Behaviors</td>
<td>Memory Clusters</td>
</tr>
<tr>
<td>Procedures, Algorithms</td>
<td>Input / Output Processors</td>
</tr>
<tr>
<td>Tasks, Transactions</td>
<td>Communications Processors</td>
</tr>
<tr>
<td>Processes</td>
<td>Switches</td>
</tr>
<tr>
<td>Threads</td>
<td>Synchronizations</td>
</tr>
</tbody>
</table>
Programmable Cellular Automata
Intermediate Formalism

Application

Intermediate
- Data Parallelism
- Cellular Robotics
- Multi-Agent Systems
- Society of Agents

Cellular Automata
- Graphs / Hypergraphs
- Attributed Structures

Architecture
Ian Foster’s Methodology:

1 Application Tasks/Process Structure/Architecture Modeling
2 Intercommunication structures ➔ Ad-hoc networks
3 Aggregation, containerization

➔ 4 Mapping, allocation, assignment to computation execution structures ➔

Map: Computational Algorithmic Structures ➔ Computer Architecture Structures
Extend to: Automatic Programming, Generative Programming
Programmable Cellular Automata

Distributed & Parallel Programming: Lots of Traditional Metaphors Available

Array Processing
Vector Processing
Image Processing
Picture Processing
Video Processing
Graphics Processing
Pattern Processing
Computational STEM
Computational Physics
Computational Chemistry
Computational Biology
Computational Math

Matrix Algebra, Calculus
Vector Algebra, Calculus
Numerical Analysis
Diakoptics, Space Filters
Network Analysis
Circuit Analysis
Tuple Programming
Data Flow Programming
CSP, CCS, CPP
Process Calculi
Process Models
OCCAM, Estelle, LOTOS

Signal Processing
DSP, VHDL
Spectral Analysis
List Processing, LISP
Sequence Analysis
Graph Algorithms
Transforms: FFT, Wavelet
Graph transformations
Array-based Languages
Computational Chemistry
Computational Biology
Computational Math

Applicative Prog.
Functional Prog.
Dataflow Prog.
Systolic alg., arch
HPC
HAS
MPI, OpenMP
BSP
Massively Parallel Proc.
Data Parallelism
Task Parallelism
Instruction Level Parallel.
Programmable Cellular Automata

Programming for Natural Parallelism

Computation = Meeting (in Space-Time) of Data & Manipulations =

Computation = \(<\) data, function / operation\> - Encounters

\(\rightarrow\) Data, Function or Both Flow to Site / Place of Computation

Manipulation of Collections of: Numbers, Codes, Symbols, Icons, Signs, Representations, Structures, Patterns, Images, Signals, Pictures, Waveforms, Tables, Arrays, Graphs, Hypergraphs, Networks, Nested Structures, Metadata, Annotated / Tagged / Colored / Attributed Structures

Alternative View: Computation is Managed by Operation System (OS)

Threads, Multi-Threading, Concurrent Programming


Programmable Cellular Automata

Data Parallelism

Algorithmic Skeletons/Kernels/Patterns
- SISD
- SIMD: Map, ApplyToAll
- SPMD
- MIMD
- Reduce: Insert
- **Kronecker Product** (KP)

Algorithmic Combinatorics
- Compositions
- Orchestration computing
- Quipu/Khipu computing
- Stream Processing
- **MapReduce** (Google)
- **Barrier Synchronization**
- Inner Product, Outer, Vector, Norm, Exterior, Distance, Metric

Applicative Programming (Functional) Style: K. Iverson, J. Backus, T. More ("I.B.M") vs. Recursive (Modern Functional Programming Style: Haskell, ML, Miranda, etc.)

Programmable Cellular Automata

Data Parallelism

\[ p = (p_1, \ldots, p_M), \quad q = (q_1, \ldots, q_N). \]
\[ A = ((a_{11} \ldots a_{M1}) \ldots (a_{M1} \ldots a_{MN})), \quad B = ((b_{11} \ldots b_{1L}) \ldots (b_{K1} \ldots b_{KL})). \]
\[ g = \text{Compose}(f_1, \ldots f_Q). \]

(SISD \( f \) \( p \)) \( \equiv \) \( f(p) \) \leftarrow \text{Use A. Church's Lambda notation}

(SIMD \( f \) \( p \)) \( \equiv \) \( (\text{Map}(f) \( p \)) \equiv ((f \ p_1) \ldots (f \ p_M))

(MIMD \( f_1 \ldots f_M \) \( p \)) \( \equiv \) \( ((f_1 \ p_1) \ldots (f_M \ p_M))

(MIMD_2 \( f_1 \ldots f_M \) \( p \)) \( \equiv \) \( ((f_1 \ p) \ldots (f_M \ p))

(\text{SPMD} \( g \) \( p \)) \( \equiv \) \( ((g \ p_1) \ldots (g \ p_M))\)
Programmable Cellular Automata

Data Parallelism

f_1(x_1) 

f_n(x_n) 

f_1(x_1) 

f_n(x_n) 

x_1 

x_n 

Barrier Synchronization
(ΣΠ-)Reduction Computing: - Aggregation, Accumulation

Data Collection = <x, y, a>; x = (x1, x2, …xN); y = (y1, y2, …, yM), N = M.
Task Collection = <f, g, h>; g = (g1, g2, …, gN); h = (h1, h2, …, hN).
Reduction Summation: Sx = (x1 + x2 + … +xN). Reduction Product: Px = (x1.x2…xN)
SP Reduction Product: SP(x, y) = (x1.y1 + x2.y2 + … + xN+yN)
Generalize: MISD(+, f, x) = (f(x1) + f(x2) + … + f(xN))
Generalize: MISD₁(f, g, x, y) = (f(…f(g1(x1, y1), g2(x2, y2)), …, gN(xN, yN)))
Generalize: MIMD₂(f, h, x, y) = (f(…f(h1(x, y), h2(x, y)), …, hN(x, y)))

J. Flynn’s Taxonomy: MISD – Multiple Instruction Single Data stream; Pipelines, Tree-based computations

Uses: Scalarization, Transforms, Convolution, FFT, Outer Product, Vector Product, Cayley Product, Matrix Product, Tensor Product, Statistics: averaging, k-moments; Scans, Prefix-sums (partial/ongoing accumulations), Max, Min

Programmable Cellular Automata

Data Parallelism:

(\Sigma \Pi -) Reduction Computing: - Aggregation, Accumulation, Pipelines
Programmable Cellular Automata

Kronecker Product

a.k.a Tensor Product, Cross Product, Cartesian Product, Gibbs-Grassmann Combinatorial Product

\[ p = (p_1, \ldots, p_M), q = (q_1, \ldots, q_N). \]
\[ A = ((a_{11} \ldots a_{M1}) \ldots (a_{M1} \ldots a_{MN})). B = ((b_{11} \ldots b_{1L}) \ldots (b_{K1} \ldots b_{KL})) \]

\[ p \times q = KP(p, q) = (((p_1 \times q_1) (p_1 \times q_2) \ldots (p_1 \times q_N)) \ldots ((p_M \times q_1) (p_M \times q_2) \ldots (p_M \times q_N))) \]
\[ A \times B = KP(A, B) = (((a_{11} \times B) (a_{12} \times B) \ldots (a_{1N} \times B)) \ldots((a_{M1} \times B) \ldots (a_{MN} \times B))) \]

Watch Out! Extensive Use Ahead! For Everyone!

KP Overdo Computations ➔ Filter / Mask / Extract / Project those of interest
➢ Symbolic (Lazy) Computational Until On-Demand of Some Calc
Compose(Map/SIMD, Reduce/Reduction)

\( p = (p_1, \ldots, p_M), q = (q_1, \ldots, q_N). \) \( f, g = \text{Compose}(f_1, \ldots, f_Q). \) \( h, \) is a binary (2-adic) operator

\( A = ((a_{11} \ldots a_{M1}) \ldots (a_{M1} \ldots a_{MN})). B = ((b_{11} \ldots b_{1L}) \ldots (b_{K1} \ldots b_{KL})) \)

Recall: \( \text{Map} (f) (p) \Rightarrow ((f \ p_1) \ldots (f \ p_M)). \) \( \text{Map} (g) (p) = ((f \ p_1) \ldots (f \ p_M)). \) \( \text{(SIMD)} \)

Recall: \( \text{Reduce} (h) (p) \Rightarrow (p_1 \ h \ p_2 \ h \ p_3 \ h \ldots \ h \ p_n) \Rightarrow (h(p_M, h( \ldots h(p_3, h(p_2, p_1))\ldots)))) \)

\( \text{(MapReduce} (f, h) (p)) \Rightarrow \text{Reduce} (h) (\text{Map} (f) (p))) \Rightarrow \text{Reduce} (h) ((f \ p_1) \ldots (f \ p_M)). \)
Programmable Cellular Automata

Composition: Quipu Orchestration

Compose(Map/SIMD, Reduce/Reduction)

\[ p = (p_1, \ldots, p_M), \quad q = (q_1, \ldots, q_N). \quad (f). \quad g = \text{Compose}(f_1, \ldots, f_Q). \quad (h), \quad h \text{ is a binary (2-adic) operator} \]

\[ A = ((a_{11} \ldots a_{M1}) \ldots (a_{M1} \ldots a_{MN})). \quad B = ((b_{11} \ldots b_{1L}) \ldots (b_{K1} \ldots b_{KL})) \]

\[ \left[\text{Compose( \{SISD, SIMD, SPMD, MIMD\}, \{SISD, SIMD, SPMD, MIMD\})}\right]^* \]

\[ = \left[\{SISD, SIMD, SPMD, MIMD\} \rightarrow \text{IntereXchange} \rightarrow \{SISD, SIMD, SPMD, MIMD\} \right]^* \]

IntereXchange:

- Flow & Inter-Communication Patterns: pt-2-pt, pt-2-mpt, mux/demux, broadcast, multicast, scatter, gather, partition, bipartite, multi-partite, MIN, switching
- Synchronization & Inter-coupling Patterns: barrier, local, global

Programmable Cellular Automata

CA Computing:
Computing over Spatial & Topological Structures of Compute Nodes

“Broadway” links
Programmable Cellular Automata

CA Computing:
Computing over Spatial & Topological Structures of Compute Nodes

Cellular Automata (CA)-like Computing: - Computing over Spatial & Topological Structures of Compute Nodes.

Node Collection = <n>; n = (n1, n2, …nK); Each node n has a neighborhood N(n) – set of direct (1-hop) neighbors. Could use Incidence Matrix & Logical Connectivity Graph repr. instead. At any time (assume discrete), each node n has a “state” <Mn, Xn, Yn> = <Internal/Memory, Input, Output>, and associated state transition rule(s).

Simplest Case (Markovian): At any time, the next state of a compute node depends on its own current state and the most recent states of the nodes in its neighborhood. Use the task collections to calculate the state transition elements, e. g., via Embarrassingly Parallel or Reduction Computations.

Generalize (non-Markovian): Neighborhood topologies of individual nodes change over time.

Generalize: Each node n has a distribution of “distance-based” multiple neighborhoods: N_{d1}(n),…,N_{dk}(n). “Mesh-of-trees” paradigm

Generalize (non-Markovian): State transitions at a node are memory-sensitive, depending on the history of the state transitions at the nodes and/or at its neighborhood(s). Systolic algorithms

Uses: Hidden Markov Models; Artificial Neuro-networks (ANN); Attribute evaluation in parse tree / syntax trees of Attribute Grammars; Cellular automata models; data flows on n-cubes, hypercubes, routing networks, multi-commodity flow networks; Traversal of topological, spatial structures and lattice structures. Data dependency models of spreadsheets. Task Parallelism; Shifts and Rotations of data; Protein folding models? Transputer/OCCAM/CSP models.

Programmable Cellular Automata
CA Computing

Fig. 2: (Locally) Synchronous cellular automaton (CA) stage [CA(n)], at time epoch n

Fig. 3: (Chaining) Sequential composition of cellular automaton stages

Programmable Cellular Automata
Flow Manipulations

<table>
<thead>
<tr>
<th>Movement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x → (S) → y</td>
<td>Relay, Sender/Receiver, Transceiver, Transponder, Bridge, Gateway, Connector, Inter-link, Inter-coupler, Portal</td>
</tr>
<tr>
<td>(x…xm) → (S) → (y1…yn)</td>
<td>Switch, Rendezvous, Mailbox, Exchange, Clearinghouse, ?Bank, Router, Permutation, Re-combinations</td>
</tr>
<tr>
<td>x → (S) → (y1…yn)</td>
<td>Broadcast, multicast, Expander, Dispatch</td>
</tr>
<tr>
<td>x → (S) → (y1…yn)</td>
<td>Demux, Scatter, Unpack, Dispatch, Queuing</td>
</tr>
<tr>
<td>(x1…xm) → (S) → y</td>
<td>Gather, Broadcast, Aggregate, Superimposition, Mux, Concentrator, Pack, Marshall, Read-All, Read-Any, Cyclic service</td>
</tr>
<tr>
<td>(x,v) ↔ (S) ↔ (y,u)</td>
<td>Synport, duplex / half-duplex channel, parallel bus</td>
</tr>
</tbody>
</table>


State-of-the-art: MPI and OpenMP;

Programmable Cellular Automata
Flow Manipulations

# Programmable Cellular Automata

**Other Intermediate Programming for Natural Parallelism**

<table>
<thead>
<tr>
<th>Interacting Units – Style 1</th>
<th>Interacting Units – Style 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA: Cellular Automata</td>
<td>OOPP: Object-oriented Parallel Programming</td>
</tr>
<tr>
<td>AN: Automata Networks</td>
<td>CSP, CCS, CPP</td>
</tr>
<tr>
<td>SAN: Stochastic Automata Networks</td>
<td>OCCAM structures</td>
</tr>
<tr>
<td>CNN: Cellular Neural Networks</td>
<td>Process Calculi</td>
</tr>
<tr>
<td>ANN: Artificial Neural Networks</td>
<td>Petri Nets, Transition Networks</td>
</tr>
<tr>
<td>DSN: Distributed Sensor Networks</td>
<td>ATN: Augmented Transition Networks</td>
</tr>
<tr>
<td>DRN: Distributed Robot Networks</td>
<td></td>
</tr>
<tr>
<td>MAS: Multi-Agent Systems</td>
<td></td>
</tr>
<tr>
<td>SOM: Society of Mind, Bots, Agents, Bots</td>
<td></td>
</tr>
<tr>
<td>Data Dependency Models (Spreadsheets)</td>
<td></td>
</tr>
<tr>
<td>Swarms, Computational Ant Intelligence</td>
<td></td>
</tr>
</tbody>
</table>
Programmable Cellular Automata

End-User Parallel Programming (EUPP): What to Expect In the Next Few Years

• Human Programmers (Non-Professional, Occasional, “End-Users”)

     +

• Computer Programming Assistants / Aides / Guides

     Human → User-oriented → Intermediate → Machine-oriented → Arch.

• Automatic Programming: Intelligent Program Translators and Transformers: Mappers, Optimizers, Tuning, Adaptors

• Affinity Computing (Anticipatory Computing)

• Transitions: Push → Pull (On Demand) → Affinity

• Cognitive Software / Affective Software, Meta-Enabled Applications

• DSL (Domain Specific Languages) vs. DSEL (DS Embedded Languages)

Programmable Cellular Automata:

Summary

Objects/Concepts ➔ Components+Containers ➔ Agent/Ants/Bots ➔ Processes/Tasks
Map: Nodes / Vertices ➔ Bots / Processes
Map: Edges / Hyper-edges ➔ Bots / Processes

(EUP) End-User Programming: Professional / Non-Professional

(DSL: Domain-Specific Languages)

Automated / Computer-Assisted Translation / Mapping

Computation Execution / Implementation
Programmable Cellular Automata:

End-User Parallel Programming (EUPP) to Map Naturally Parallel Applications Into Massively Parallel Computer Hardware Architectures

Thank You