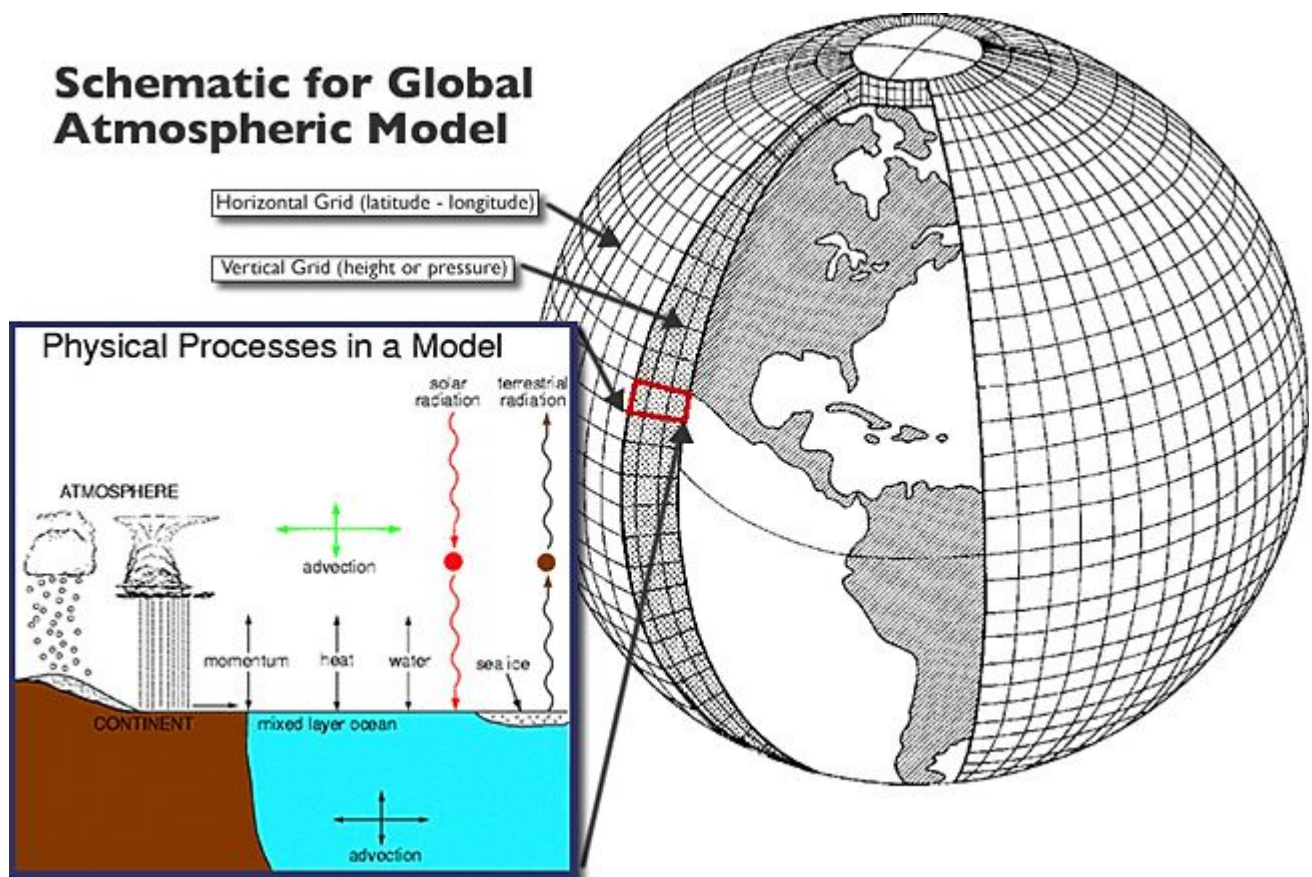


“Grand Challenge” Problems : Computational Climate Prediction

A possible solution :

A Low Complexity Dynamic Grid Coprocessor
for Distributed Computing



Do you see any design, logic or any other kind of error?

Contact

“Climate Prediction CPU”

dist23 at juno dot com

Numerical weather and climate prediction uses known weather conditions as baseline inputs to extrapolate “short term” weather forecasts and “long term” climate outcomes.

Initial weather and climate conditions are fed into an array of simplified “ensemble models” of the atmosphere, typically with only a few minor value changes between hundreds of models. Due to chaos theory, the slightest change in a model's input parameters typically yields hundreds if not thousands of vastly different outcomes.

Large numbers (dozens if not hundreds) of climate and weather simulations must be run (and averaged) to filter or “sort out” unrealistic variance. Use of model ensemble forecasts helps to define the forecast uncertainty and extend weather and climate predictions farther into the future than would otherwise be possible.

Minor changes in model inputs and outputs can be used to extrapolate historical climate conditions (called hindcasting).

Manipulating these huge datasets [as well as performing the complex prediction calculations necessary at a resolution high enough to make the results useful] requires very powerful computers. Supercomputers are ideal for this task, but very few supercomputers are available to use for this task.

History

British mathematician Lewis Fry Richardson first proposed numerical weather prediction in 1922, although his initially unsuccessful attempts at the method date back to WWI. Richardson attempted to perform many kinds of low complexity numerical forecasts before World War II, but success was never achieved.

The first successful numerical prediction was performed in 1950 by a team composed of the American meteorologists Jule Charney, Philip Thompson, Larry Gates, and Ragnar Fjörtoft and applied mathematician John von Neumann, using the ENIAC digital computer.

They used a simplified form of atmospheric dynamics based on the barotropic vorticity equation. This simplification greatly reduced demands on computer time and memory, so that the computations could be performed on the relatively primitive computers available at the time. Later models used more complete equations for atmospheric dynamics and thermodynamics.

Operational numerical weather prediction began in 1955 under a joint project by the US Air Force, Navy, and the US Weather Bureau (later NOAA).

How is modern numerical forecasting done?

Numerical atmosphere models are initialized using data collected from radiosondes, weather satellites, and surface weather observations. These irregularly-spaced observations are processed by data assimilation and objective analysis tools which perform quality control checks that filter out misbehaving sensors.

Guessimate data values are extrapolated for locations that are physically unreachable by the sensor net. Extrapolated data points (due to corrupt sensors or lack of data) are imputed using known and expected seasonal variances.

The so called atmospheric “primitive equations” are then applied to the atmospheric state to find existing and new rates of change. The known rates of change predict the atmosphere in a future state. Time stepping procedures are continually repeated until the solution reaches the desired forecast time.

The length of the time step is related to the distance between the points on the computational grid. Time steps for global climate models may be on the order of tens of minutes, while time steps for regional models may be 30 seconds to 30 minutes.

Longer term problems seeking solutions

Most climate models simulate a region of the Earth's atmosphere from the surface to the Stratopause. There also exist numerical models which simulate the wind, temperature and composition of the Earth's tenuous upper atmosphere, from the mesosphere to the exosphere, including the ionosphere.

This region is affected strongly by the 11 year Solar Cycle through variations in solar UV / EUV / X-ray radiation and solar wind leading to high latitude particle precipitation and aurora.

It has been proposed in the overall body of atmospheric sciences research – that these high energy phenomena may have a significant effect on the lower atmosphere. There are proposals on how high energy forcing should be included in simulations of climate change, but none have reached the applications used in Distributed Computing.

For this reason there has been a drive in recent years to create "whole atmosphere" models to investigate whether or not this is the case.

The atmosphere as well as the world's oceans (that influence and buffer atmospheric trends) are dynamic fluids. The basic idea of numerical weather prediction is to sample the state of the fluid at a given time and use the equations of fluid dynamics and thermodynamics to estimate the state of the fluid at some time in the future.

One strange defect of every climate simulation that has a model time of more than a years – is that the Gaussian Second is not accounted for at all. The Gaussian Second is important with respect to solar forcing, and overall atmospheric behavior with respect to the Earth gradually slowing down. Some 150 million years ago, the day was 23 hours – not the 24 hours known in the current era.

In deep time (before the Jurassic) climate models not only need to account for a faster planet but a weaker Sun.

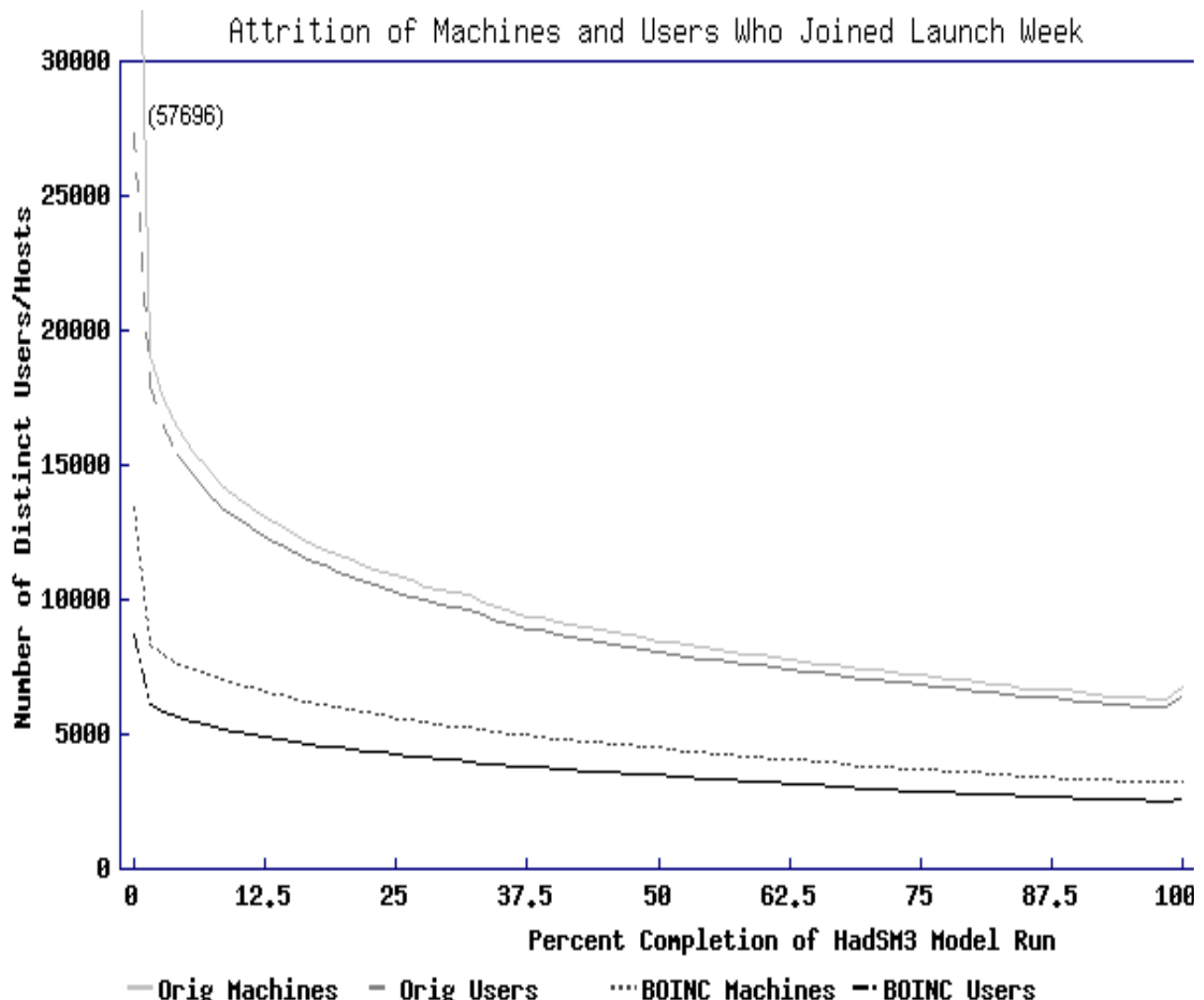
Contrasts (goals)	NWPs (predict weather)	GCMs (predict climate)
Spatial coverage	Regional or Global	Global
Temporal range	3 Days to 25 Weeks	2 Years to 15 Decades
Spatial resolution	Variable (20-100 km ²)	Usually ≥ 200 km ²
Relevance of initial conditions	High	Medium to Low
Relevance of clouds, radiation	High	Medium
Relevance of surface Land, Ice, Ocean...	Medium to Low	High
Relevance of ocean dynamics	Low	High
Relevance of model stability	Medium to Low	High
Time dimension	Essential	For comparison with other model runs
Time Step	2 minutes to 20 minutes	30 minutes to 60 minutes
Physics	Equations of motion (plus radiative transfer equations, water conservation equations ..)	
Methods	Finite difference expression of continuous equations, or spectral representation; run prognostically	
Maximum time step	Controlled by spatial resolution (CFL condition)	

THE CURRENT “Climate Simulation” MESS

The problem with current climate simulation [and fluid dynamics computing technology in general] is that most non-University users are not able to easily complete their simulations (or work units) with the current Climate Simulation programs available on Personal Computers (PCs).

The attrition rate with PCs runs at ~95% and has not improved substantially in the past 5 years.

Supercomputers do this task much better than home computers – but there are too few supercomputers available to be dedicated to climate prediction.



This high attrition rate (aka failure rate) has very little to do with available RAM memory on these machines (PCs usually have 1 Gigabyte as a minimum, but 4 Gigabytes is typical as of 2014).

The extremely high computational burden of running a mixed scalar / vector / vorxel block of data is bad enough. The problem is that these models have a high internal complexity – and consume substantial disk space.

The current “State of the Art” is not acceptable.

The current mixed C++11 and FORTRAN codebases (using vector and scalar code) are not helping climate prediction applications used in distributed computing – ADA should be the mandatory choice for creating reliable climate simulations.

This table is designed to give one some idea of the telecommunications system costs of computing discrete vorxels over the Internet. “Telecom Cost” relates to reading nearby neighbors “vorxel record” values. “System Telecom Cost” is based on telecom system IO costs for reading and writing vorxel values to neighbors. This table is only intended to be approximate -- the math is not perfect.

Vorxels (1, 4, 8, 9...)	Neighbors				Telecom Cost	System Telecom Cost
1	N1		N2		N3	
	N4		V		N5	
	N6		N7		N8	
	8 neighbors/vorxel					
4	N01		N02		N03	
	N04		N05		N06	
	N07		V1		V2	
	N08		V3		V4	
	N09		N10		N11	
N12						
{(8-3) neighbors/vorxel} x 4						
8 (in the form of a 2 x 4 array)	N01		N02		N03	
	N04		N05		N06	
	N07		V1		V2	
	N08		V3		V4	
	N09		V5		V6	
N10		V7		V8		
N11		N12		N13		
N14		N15		N6		
{(8-3) neighbors/vorxel} x 4 + {(8-5) neighbors/vorxel} x 4						
8 “non local” vorxel reads per computation, but no local vorxel reads					(8 _N x 2) = ~16, edge vorxel mapping cost ~4%	
20 “non local” vorxel reads per computation, but 12 “local reads”					(40 _N – 24 _L) = ~16, edge vorxel mapping cost ~5%	
(20 + 12) = 32 “non local” reads per computation but (12 + 20) = 32 “local reads”					(32 _N - 32 _L) = ~0, edge vorxel mapping cost ~6%	

The best case is a Telecom Cost of ~32 at a System Cost of ~ 0 (but a telecom bill of ~64).

I have not factored in “9 x 9” to “17 x 17” globs of vorxels, but using globs of vorxels this numerous is for all practical purposes a different kind of modeling.

You cannot win by having each client compute a single vorxel (the telecommunications costs are too high), but you can partly win by having each client compute a few more vorxels than just one.

Many more vorxel read-write cycles are required in these models than what one might initially suspect – unless the computational mesh describes a simple system like a static electrodynamic system (not a lot of electrodynamic motion) or airflow over a wing surface (predictable physical dynamics).

Making vorxels pentagonal (5 sided), hexagonal or octagonal (8 sided) will not fix the problem either.

You can successfully do fluid dynamics computations over the Internet by distributing your vorxels across multiple CPUs, but the overall telecoms cost quickly become stratospheric.

It goes without saying that your simulation will always have ongoing synchronization issues if you do not properly use data compression and several layers of checksums – and scaling your model to different resolutions may be problematic if you don't object orient (for more than one resolution) your model's data structures.

Current Models and Modeling

So, simulate the entire system on your computer – that will work, right?

There are about about 5 “climate prediction” distributed computing applications on the Internet as of 2008. Not one of these apps uses the distributed “groups of voxels” approach, in spite of its possible advantages.

The “groups of voxels” approach is implemented in a very limited way, but not for climate prediction. Only for low complexity distributed fluid dynamics applications use this approach – and these apps have a very low profile and equally low levels of success.

Models currently deployed

HadCM3 (Hadley Centre Coupled Model, Version 3) is a coupled atmosphere-ocean general circulation model (AOGCM) developed at the Hadley Centre in the United Kingdom.

Unlike earlier AOGCMs at the Hadley Centre and elsewhere (including its predecessor HadCM2), HadCM3 does not need flux adjustment (additional "artificial" heat and freshwater fluxes at the ocean surface) to produce a good simulation.

The higher ocean resolution of HadCM3 is a major factor in this; other factors include a good match between the atmospheric and oceanic components; and an improved ocean mixing scheme. HadCM3 has been run (in a steady state mode) for over a thousand years, showing little non-seasonal drift in its surface climate.

HadCM3 is composed of two components: the *Atmospheric Model* “*HadAM3*” and the *Ocean Model* “*HadOM3*” (which includes a sea ice model).

Atmospheric model (HadAM3)

HadAM3 is a grid point model and has a horizontal resolution of $2.5^\circ \times 3.75^\circ$ (Latitude x Longitude). This gives 7008 data points (from a 96×73 grid) on the scalar (pressure, temperature and moisture) grid; the vector (wind velocity) grid is offset by $1/2$ a grid box.

- This gives the HadAM3 an overall horizontal resolution of approximately 300 km^2 .
- There are 19 levels in the HadAM3 vertical model (elevation domain).

The timestep is 30 minutes (with three sub-timesteps per timestep in the dynamics). Near the poles, fields are Fourier-filtered to prevent instabilities due to the Courant-Friedrich-Levy (CFL) criterion and conditions.

Ocean model (HadOM3)

The ocean model has a resolution of $1.25^\circ \times 1.25^\circ$ and a timestep of 1 hour, and can be coupled to the Atmospheric Model at either the same update rate or at lower update rates (typically in force with the current distributed computing models). There are 6 ocean grid points for every atmospheric one. For ease of coupling the two models the grids are aligned and the ocean coastline is forced to be aligned to the atmospheric grid.

HadCM3 Coupling

The atmospheric model is run for a day, and the fluxes (of heat, moisture and momentum) at the atmosphere-ocean interface are accumulated. Then the ocean model is run for a day, with the reverse fluxes accumulated. This then repeats through the length of the run.

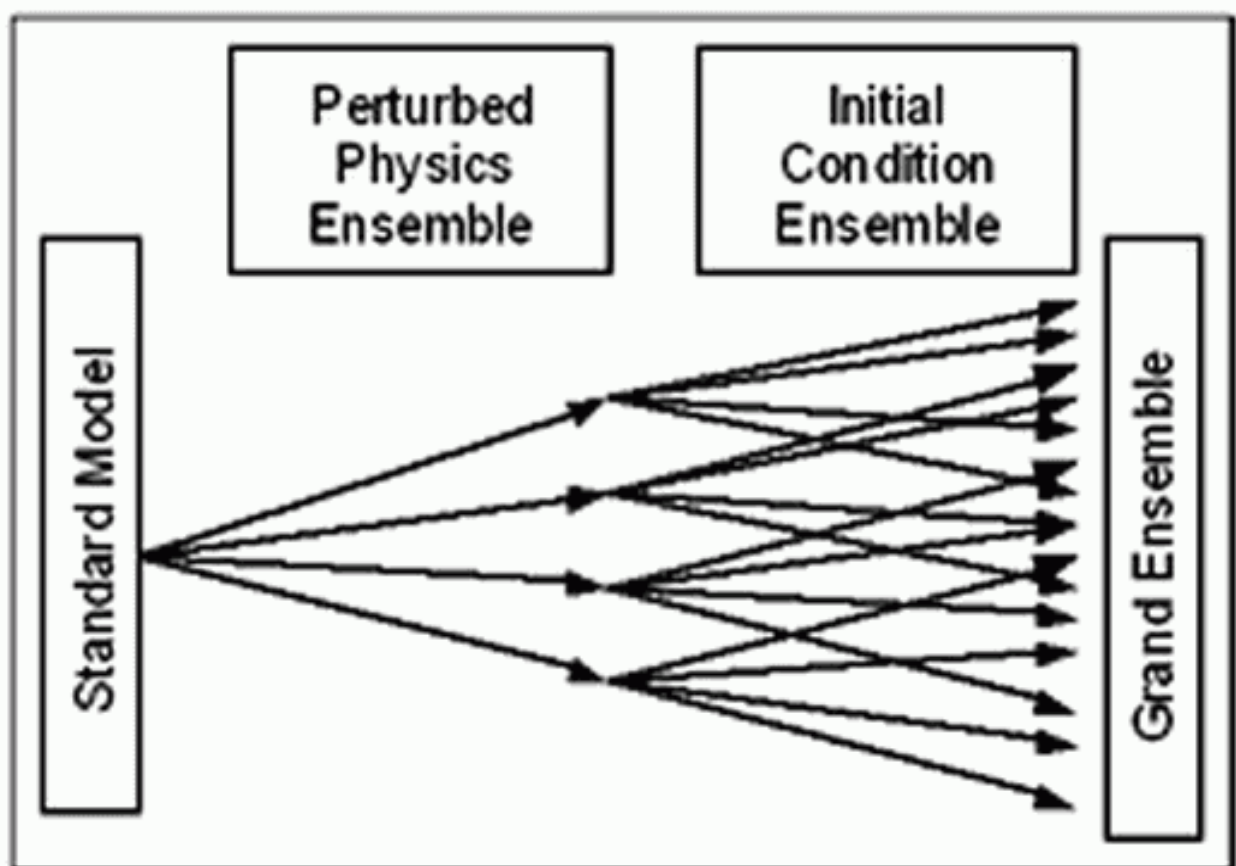
The once per day coupled update rate may be too slow considering this self correcting model's capabilities. It would be reasonable to suggest that ~7 hour intervals for flux calculation and correction are possible.

The ocean model incorporates a thermodynamic-dynamic sea ice model with a primitive (ocean drift) dynamics.

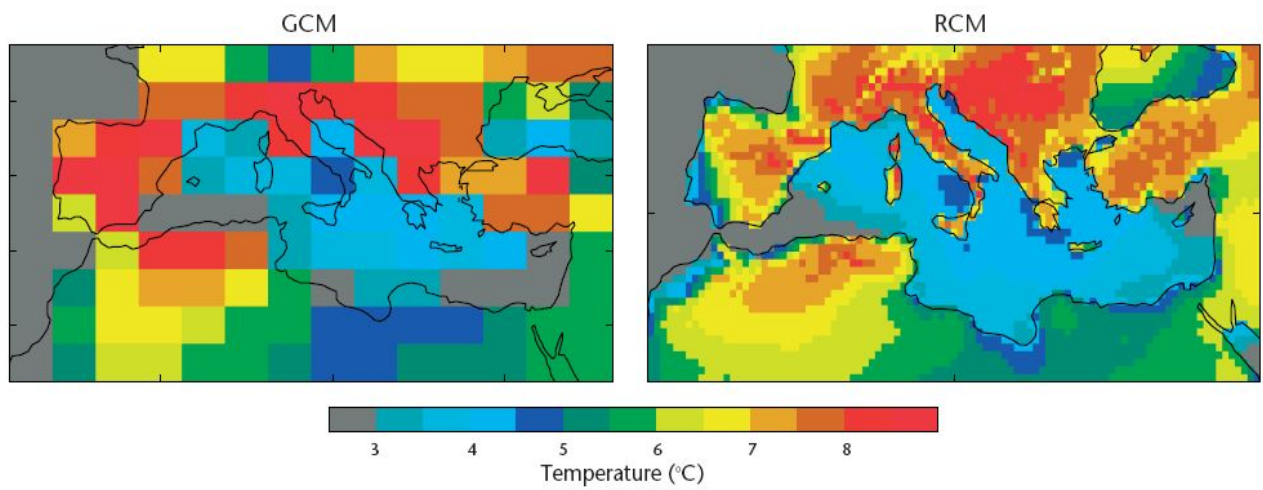
CHIME

The Coupled Hadley-Isopycnic Model Experiment (CHIME) is a new coupled model, developed at Southampton Oceanography Centre. It consists of the atmospheric model used in the Hadley Centre's HadCM3 climate model, coupled to a hybrid coordinate ocean model (HYCOM) via the OASIS coupler. The ocean model grid is identical over most of the globe to that used in HadCM3, allowing the effect of the different vertical representation of the ocean to be easily assessed.

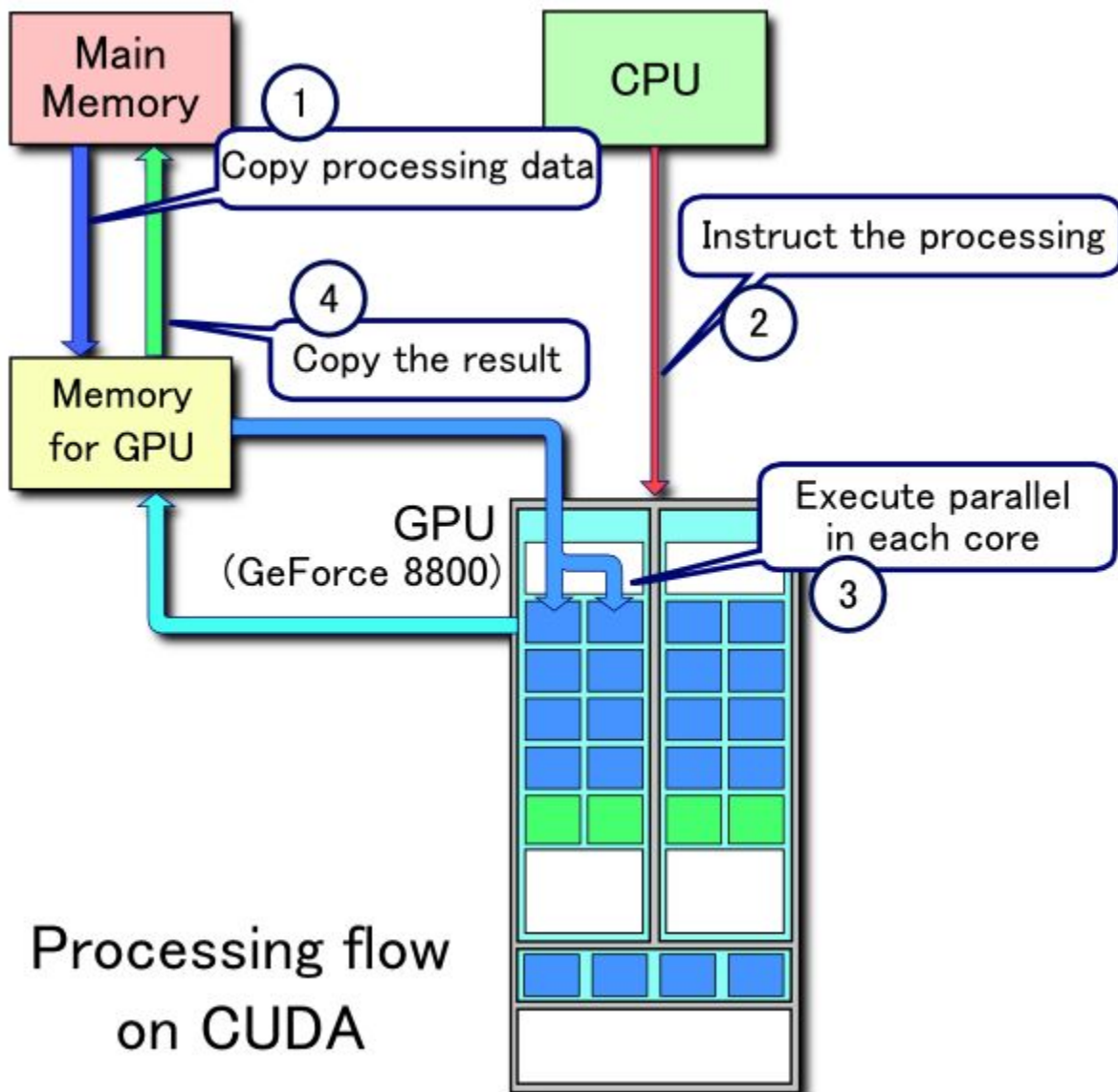
A further innovative feature of CHIME's ocean component is its use of a bipolar grid over the Arctic. This avoids the problem of convergence of meridians toward the North Pole by replacing the latitude-longitude grid north of a given latitude circle (of 55° N) with a bipolar grid. This matches the spherical grid perfectly at this latitude, but its two poles may be placed harmlessly in large land masses. At this point CHIME has not been deployed in distributed computing yet, but could in future due to its close design relationship with HadCM models.



A problem in search of a CPU



A comparison between global and regional resolution in climate modeling



Example of CUDA processing flow

1. Copy data from main memory to GPU memory
2. CPU instructs the process to GPU
3. GPU execute parallel in each core
4. Copy the result from GPU memory to main memory

The CUDA approach, although interesting in its own right -- is not quite good enough for generalized Climate Prediction programs.

Although CUDA coprocessors may be perfect as a coprocessor, the CUDA coprocessor lacks the internal design infrastructure for vorxel processing.

CUDA's lack of massive coprocessor computing unit “hyper” parallelism renders it a poor candidate for vorxel computing.

Flynn's taxonomy

Michael J. Flynn created one of the earliest classification systems for parallel (and sequential) computers and programs, now known as Flynn's taxonomy. Flynn classified programs and computers by whether they were operating using a single set or multiple sets of instructions, whether or not those instructions were using a single or multiple sets of data.

The single-instruction-single-data (SISD) classification is equivalent to an entirely sequential program. The single-instruction-multiple-data (SIMD) classification is analogous to doing the same operation repeatedly over a large data set.

This is commonly done in signal processing applications. Multiple-instruction-single-data (MISD) is a rarely used classification. While computer architectures to deal with this were devised (such as systolic arrays), few applications that fit this class materialized. Multiple-instruction-multiple-data (MIMD) programs are by far the most common type of parallel programs.

Instruction-level parallelism

A computer program is, in essence, a stream of instructions executed by a processor. These instructions can be re-ordered and combined into groups which are then executed in parallel without changing the result of the program. This is known as instruction-level parallelism. Advances in instruction-level parallelism dominated computer architecture from the mid-1980s until the mid-1990s.

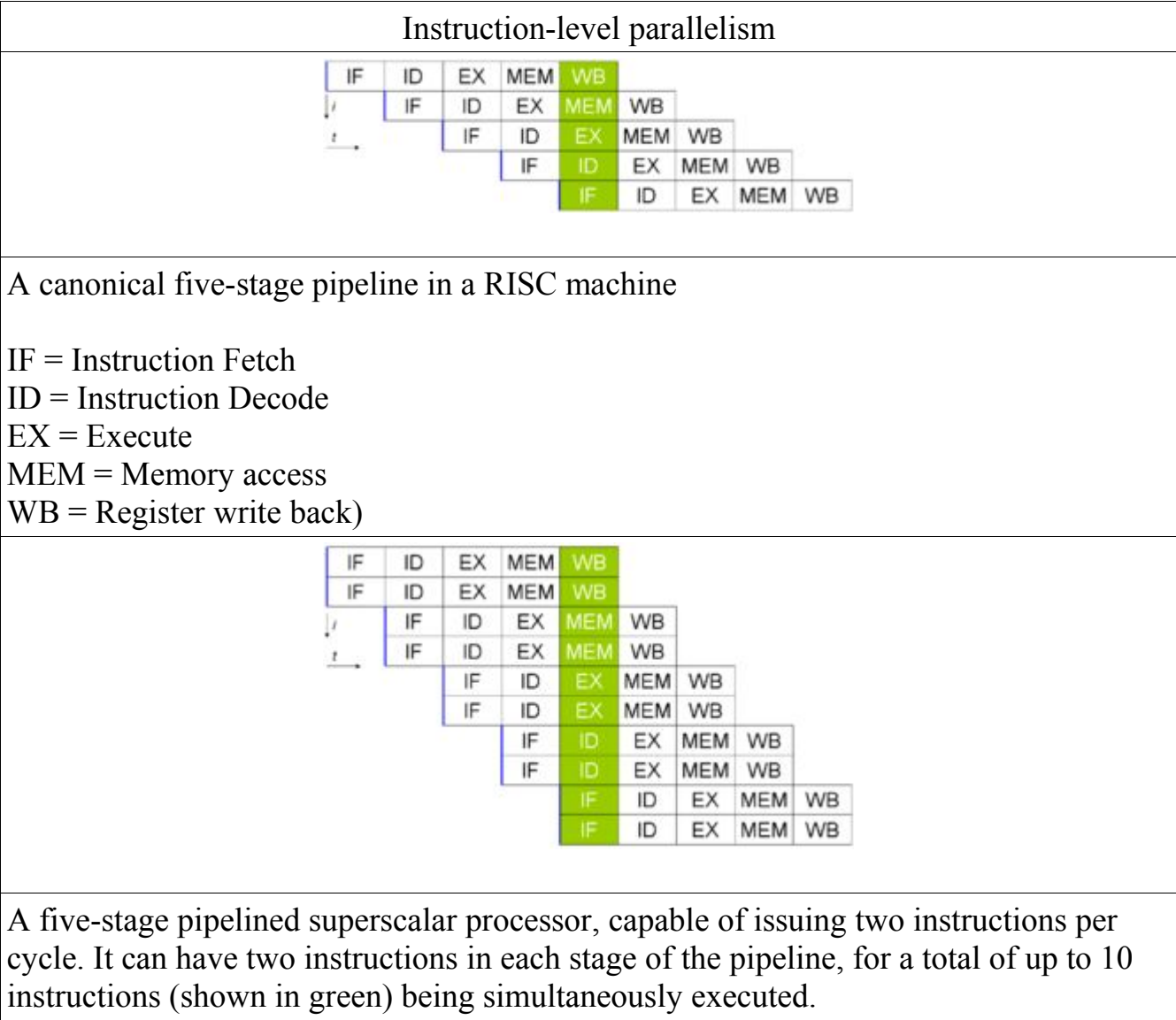
Modern CPUs have multistage instruction pipelines. Each stage in the pipeline corresponds to a different action the processor performs on that instruction in that stage; a processor with an N-stage pipeline can have up to N different instructions at different stages of completion.

The canonical example of a pipelined processor is a RISC processor, with five stages: instruction fetch, decode, execute, memory access, and write back. The Pentium 4 processor had a 35-stage pipeline, a highly complex pipeline.

In addition to instruction-level parallelism from pipelining, some processors can issue more than one instruction at a time. These are known as superscalar processors. Instructions can be grouped together only if there is no data dependency between them.

Scoreboarding and the Tomasulo algorithm (which is similar to scoreboarding but makes use of register renaming) are two of the most common techniques for implementing out-of-order execution and instruction-level parallelism.

Scoreboarding is recommended to keep gate complexity low.



Noteworthy application specific coprocessors

D-Wave was spun out of the University of British Columbia (UBC) in 1999 to commercialize superconductor-based, quantum computer processors.



Quantum computers (QCs) use quantum mechanics (QM), the rules that underlie the behavior of all matter and energy, to accelerate computation. It has been known for some time that once some simple features of QM are harnessed, machines will be built capable of outperforming any conceivable conventional supercomputer. QCs are not just inherently faster than conventional computers. They change what computer scientists call the computational scaling of many problems.

In 1936, mathematician Alan Turing published a famous paper that addressed the problem of computability. His thesis was that all computers were equivalent, and could all be simulated by each other. By extension, a problem was either computable or not, regardless of what computer it was run on. This paper led to the concept of the Universal Turing Machine, an idealized model of a computer to which all computers are equivalent.

As an example, consider the modeling of a nanometer sized structure (such as a drug molecule) using conventional (non-quantum) computers.

Solving the Schroedinger Equation (SE), the fundamental description of matter at the QM level, more than doubles in difficulty for every electron in the molecule. This is called exponential scaling, and prohibits solution of the SE for systems greater than about 30 electrons.

For example: The typical Caffeine molecule has 100+ electrons. This property makes it roughly “almost a Googleplex (10^{100}) harder” a quantum system to solve by conventional means.

In a written out form a Caffeine atom is some
~100,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000
times more difficult to solve than a typical 30-electron system.

As of the late-2000s most 30 electron quantum systems make most high-end supercomputers choke or outright fail.

This restriction makes first-principles modeling of molecular structures impossible, and has historically defined the boundary between physics (where the SE can be solved by brute force) and chemistry (where it cannot, and empirical modeling and human creativity must take over).

Quantum computers are capable of solving the SE with linear scaling exponentially faster and with exponentially less hardware than conventional computers. For a QC, the difficulty in solving the SE increases by a small, fixed amount for every electron in a system. Even very primitive QCs will be able to outperform supercomputers in simulating nature.

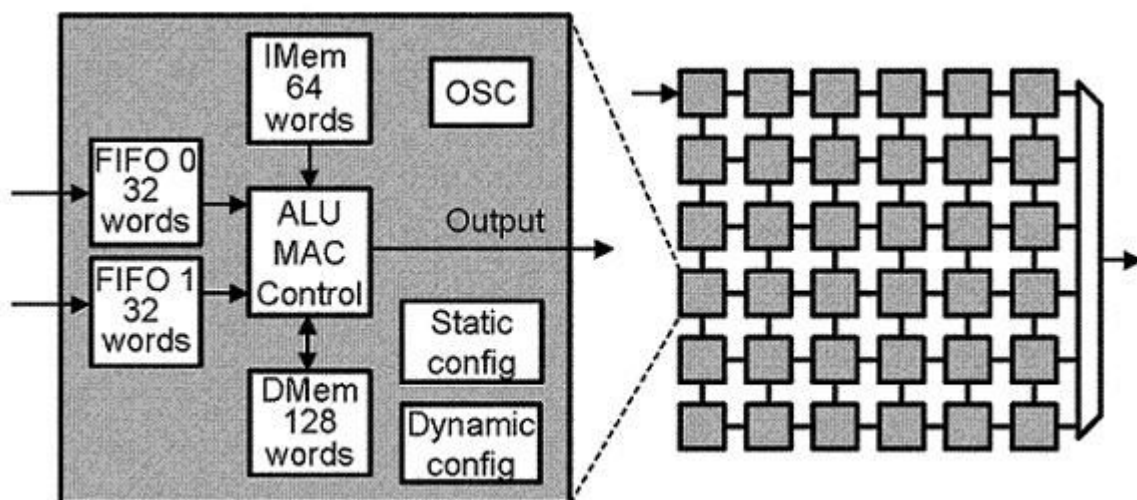
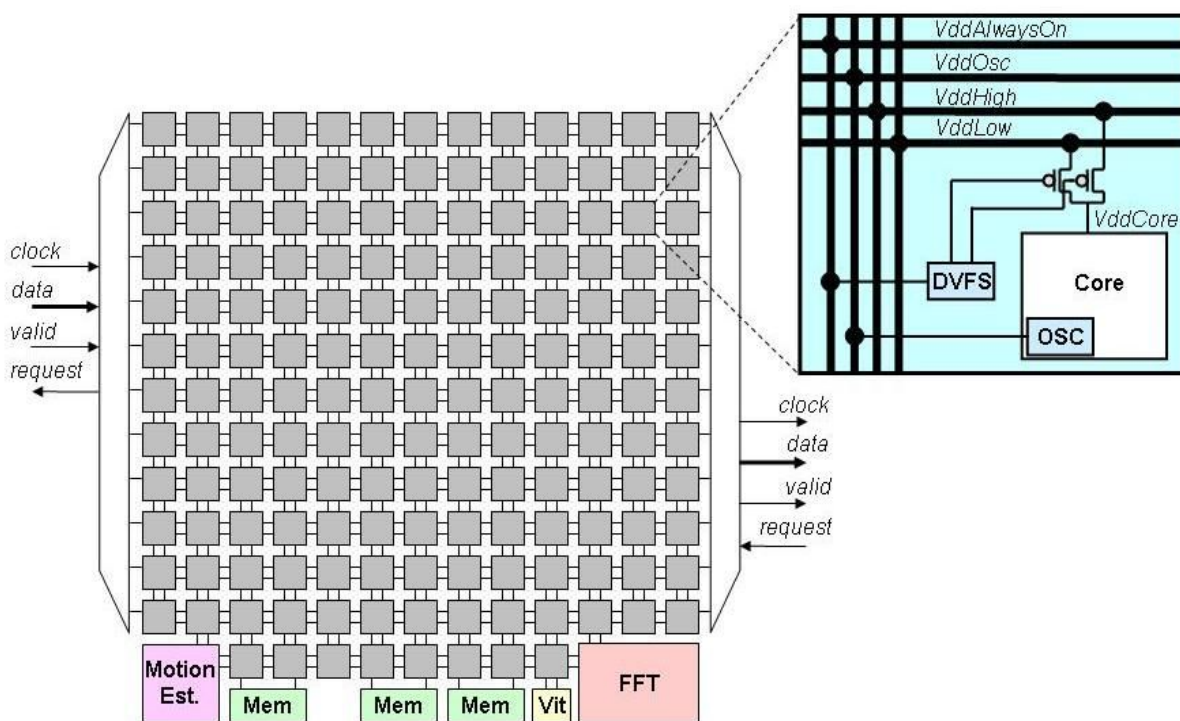
Noteworthy generalized coprocessors

Asynchronous Array of Simple Processors (AsAP) project

This single-chip processing system is comprised of a large number of fine-grain asynchronously-operating programmable processors connected by a reconfigurable 2-dimensional mesh network.

This is a 0.18 μm CMOS chip that was fabricated during the Summer of 2005. Current data suggests that this is the highest clock rate fabricated processor designed in any university.

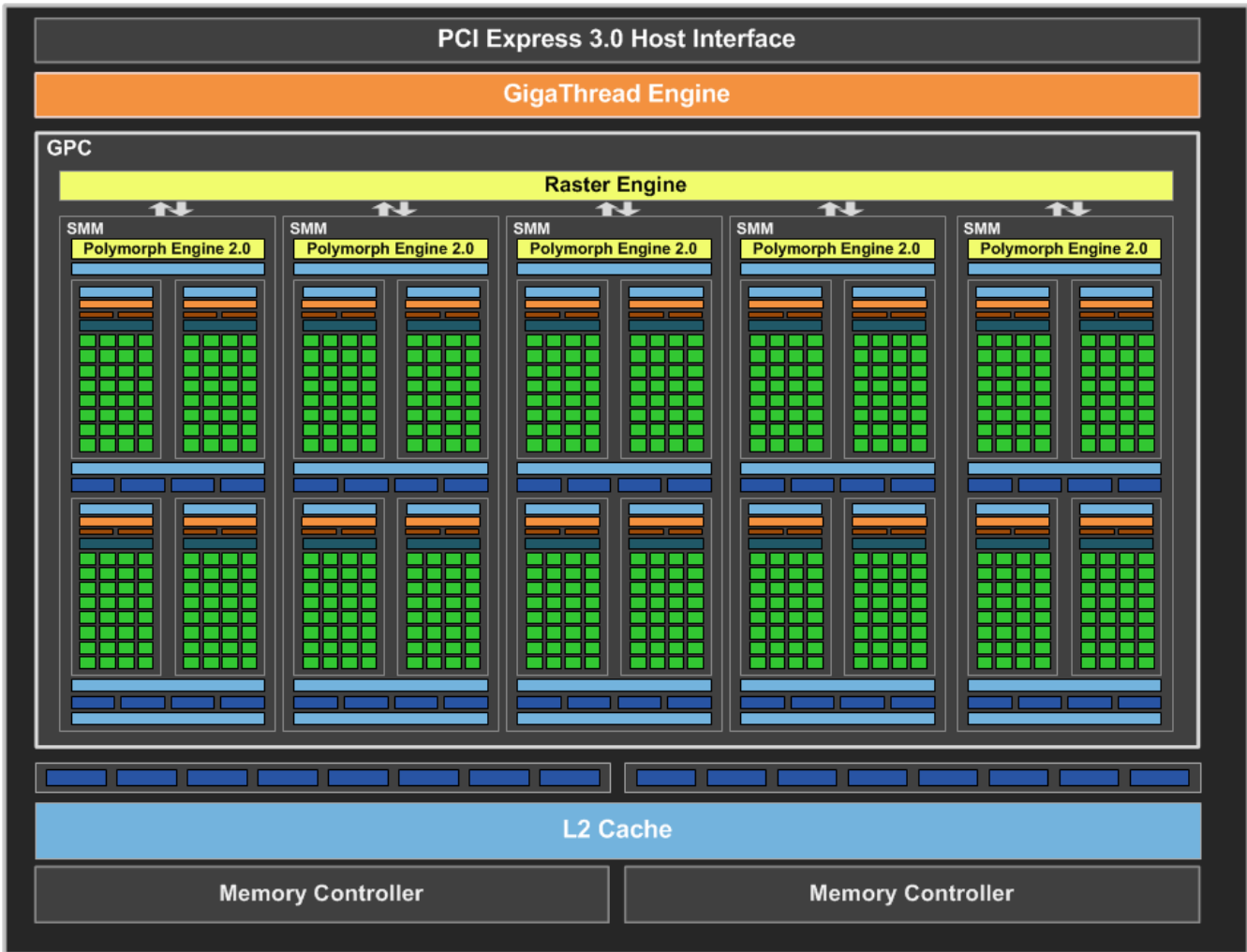
A 13 mm x 13 mm chip utilizing the exact same design in 90 nm CMOS would contain more than 1000 processors and be capable of more than 1 Tera-op/sec peak performance.



Another possible CPU solution, originating from the Game Console trade

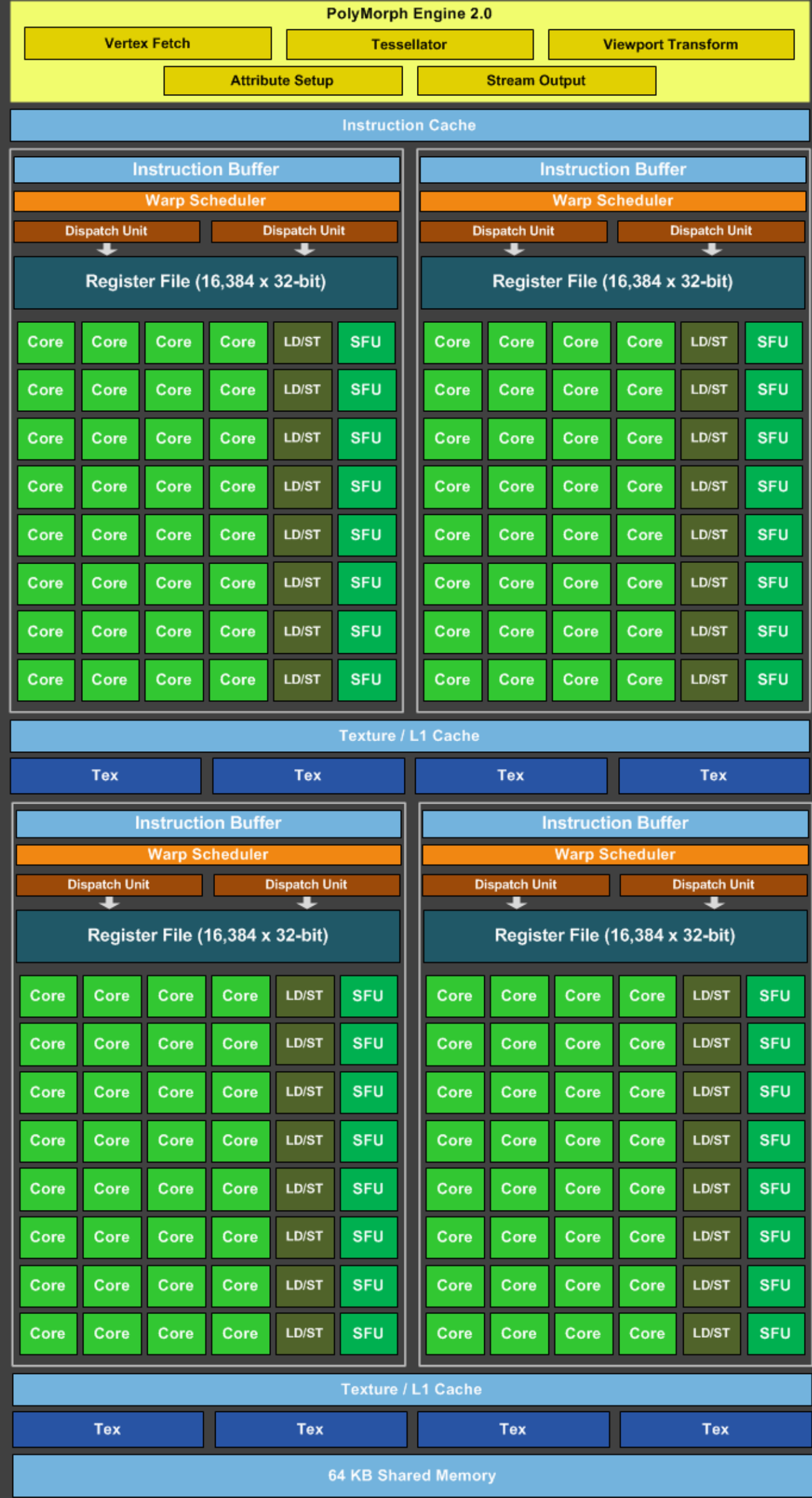
The GM107 GPU contains one GPC, five Maxwell Streaming Multiprocessors (SMM), and two 64-bit memory controllers (128-bit total).

This is the full implementation of the chip, and is the same configuration we ship with the GeForce GTX 750 Ti.



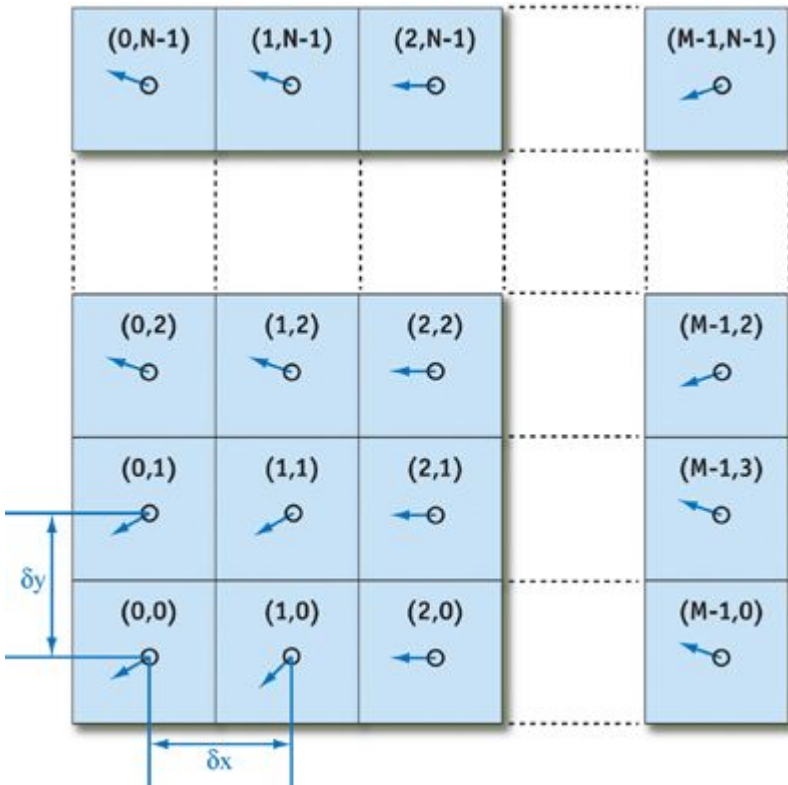
GPU	GK107 (Kepler)	GM107 (Maxwell)
CUDA Cores	384	640
Base Clock	1058 MHz	1020 MHz
GPU Boost Clock	N/A	1085 MHz
GFLOPs	812.5	1305.6
Texture Units	32	40
Texel fill-rate	33.9 Gigatexels/sec	40.8 Gigatexels/sec
Memory Clock	5000 MHz	5400 MHz
Memory Bandwidth	80 GB/sec	86.4 GB/sec
ROPs	16	16
L2 Cache Size	256KB	2048KB
TDP	64W	60W
Transistors	1.3 Billion	1.87 Billion
Die Size	118 mm ²	148 mm ²
Manufacturing Process	28-nm	28-nm

SMM



The Climate Prediction Coprocessor

General Proposal A2



Climate Prediction Coprocessor: Design Goals & Philosophy (overview)

Create a 100 MHz coprocessor that is capable of running a global climate simulation at a resolution of $1^\circ \times 1^\circ$ (or less, $\sim 360^2 = 129,600$ voxels) with a complete model update rate of 1 Hz.

A computable voxel here is considered to be something like

```
Record { real data_current[128]; real data_previous[128]; } // Real = 32 bit signed real number
```

The USB or Firewire coprocessor subsystem “add on” should have a target consumer cost of 60 EUR, with a fabrication cost for the chip itself of 20 EUR.

The coprocessor must have mixed RISC / DSP nature, but creeping CISC instruction set features should be avoided “at all costs” where possible. This CPU must be as simple as possible so that it can be duplicated in mass in Silicon, or possibly Gallium Arsenide.

The coprocessor must also be able to offer 1 Hz model updates for simulations of

- magnetic core electrodynamics of the Earth
- fluid dynamics processes of Jupiter, Saturn etc...
- electrodynamic processes of Jupiter, Saturn, etc...

The coprocessor must be able to achieve this speed regardless of cell (voxel) complexity, linearity, temporal resolution or spacial resolution.

The processor must enable an 180 year climate model to be run in 10 days, with the possibility of a 360 year climate model being completed in 25 days.

The coprocessor must also be capable of code breaking functionality and suitable for hash function collision research.

- SHA1 collision testing should be within the core capability of the coprocessor
- Enigma @ Home should be implementable at the individual coprocessor level

The coprocessor must have an API, allowing for continuous self testing and scheduling of the individual processors. The calling program should not need to be highly knowledgeable about the nature of the CPU itself.

Climate Prediction Coprocessor: Design Goals & Philosophy

The logic describing the Coprocessor should

- be maintained in open source VHDL or Verlog
- be openly viewed by the general public for design flaws
- be Open Source, but any major changes tightly overseen by an architecture committee

The coprocessor chip itself

- should be a computer add on, attachable via a USB or Firewire port.
- must be energy efficient, consuming typically under 25 watts.
- must have high internal throughput with as many internal parallel asynchronous data lines between its internal computational and memory subsystems as possible.
- must support a Coprocessor Mapping Unit that can logically map coprocessors and memory for the controlling CPU (like a Power PC)

The coprocessors internal architecture -- for each coprocessor it

- must support signed 32 bit floating numbers.
- must support signed 32 bit integers.
- must support unsigned 32 bit integers.
- must support an ID facility to indicate coprocessor version and subversion.
- must forbid self modifying code in any form by design.
- must not support register renaming, as it creates unnecessary complexity.
- must support a 16 bit memory space, 32k for code and 32 k for data.
- must support memory mapping to a 32 bit address space.
- must support a master watchdog timer of 40 bits for each coprocessor, if it runs down to zero the individual coprocessor must alert the coprocessor controller.
- must support Saturation Arithmetic, in which operations that produce overflows will accumulate at the (maximum or minimum) values that the register can hold rather than wrapping around (maximum+1 never yields minimum, and minimum-1 never yields maximum).
- should have a unique ID Number for each individual coprocessor, to track its health over its lifetime.
- should have a unique Version Number, to track its health over its lifetime.
- should support limited Out of Order Processing, but only in later versions.

The coprocessor instruction set

- must support 32 physical constants (physics, chemistry)
- must support Pi, e
- must use only 16 bit instructions.
- should average 2 to 4 clock cycles, except for: NOP, RESET, JUMP...

The coprocessor typically would

- be implemented at the density of instances 300 per Coprocessor unit, possibly 3000 in future.
- be controlled by a Power PC 601 Core, or whatever is also used by

The Climate Prediction GPU Instruction Set *must support*

Instruction	40 bit float	40 bit signed int	Total # of Instructions
RESET REG %/# (+0, +0.0)	x	x	1 (supervisory)
NOP	NA	NA	2
CALL or JUMP	NA	NA	1
RETURN	NA	NA	2
JUMP EXCEPTION	x	x	3
GREATER THAN	x	x	1
LESS THAN	x	x	2
EQUAL	x	x	3
Scientific or Math Constant (128)	x	NA	1
AND (1)	NA	x	1
NAND (1)	NA	x	2
OR (1)	NA	x	3
NOR (1)	NA	x	4
XOR (1)	NA	x	5
NOT (1)	NA	x	6
Rotate Left (1)	NA	x	1
Rotate Right (1)	NA	x	2
Shift Left (1)	NA	x	3
Shift Right (1)	NA	x	4
ADD	x	x	1
SUBTRACT	x	x	2
MULTIPLY	x	x	3
DIVIDE	x	x	4
VECTOR ADD	x	NA	1
VECTOR SUBTRACT	x	NA	2
VECTOR MULTIPLY	x	NA	3
VECTOR DIVIDE	x	NA	4

(1) Instruction must operate in “full register mode” and “byte mode”.

The Climate Prediction GPU Instruction Set *must support*

Instruction	40 bit float	40 bit signed int	Total # of Instructions
Sin(x) [degrees]	x	NA	1
Cos(x) [degrees]	x	NA	2
Log_10(x)	x	NA	3
Exp(x)	x	NA	4
Abs(x)	x	x	5
Inverse	x	NA	6
DCT-JPEG (1)	NA	x	1
FFT32 (1)	NA	x	2
FMA (1,2)	x	x	3
Float32 to Int32	x	NA	1
Int32 to Float32	NA	x	2
Float32 to Float40	x	NA	3
Float40 to Float32	x	NA	4

- (1) Registers not used, only memory. Output is to registers however, still working on implementation idea here. Float in syntax is also know as 'fused' in DSP nomenclature. DCT-JPEG will ignore the sign bit.
- (2) CPUs have been following the IEEE Standard for Floating-Point Arithmetic, also known as IEEE 754 standard, for quite some time: the original standard was published in 1985, and was updated to IEEE 754-2008 in 2008.
- This standard made it possible to write algorithms using floating-point arithmetic, which could be executed, on a variety of platforms with identical results. One of the main additions was the introduction of a Fused Multiply-Add (FMA) instruction.
- When implemented in hardware, the equivalent instruction takes about the same time as a multiply, resulting in a performance advantage for many applications.
- While an unfused multiply-add would compute the product $a \times b$, round it to P significant bits, add the result to c, and round back to P significant bits, a fused multiply-add would compute the entire sum $(a \times b + c)$ to its full precision before rounding the final result down to P significant bits.
- The latest generation of NVIDIA GPUs, like the Tesla C2050, has support for the IEEE 754-2008 FMA both in single and double precision. It is possible to disable generating this instruction in CUDA FORTRAN using a compiler flag “-Mcuda=nofma”

The Climate Prediction GPU Instruction Set *must support*

Exceptions	40 bit float	40 bit signed int	Exception #
Division by 0	x	x	1
Division by Infinity	x	x	2
Multiplication by 0	x	x	3
Multiplication by Infinity	x	x	4
Memory Read Fault (1)	x	x	5
Memory Write Fault (1)	x	x	6
Bus Read Fault	NA	NA	7
Bus Write Fault	NA	NA	8

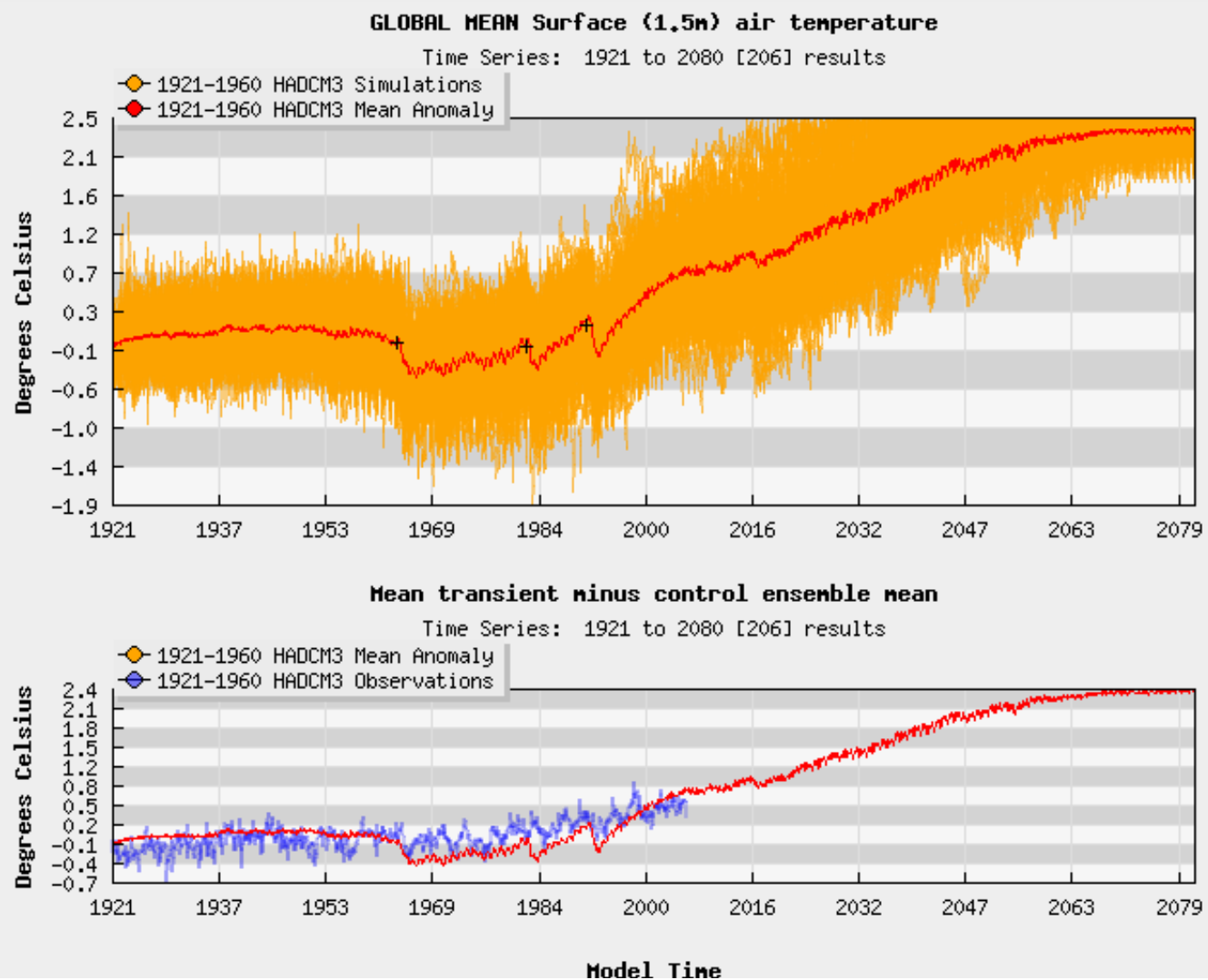
(1) Applies only to local memory used by the coprocessor, scope is thus only 64 Kb. Useful for tagging coprocessors that should be excluded from the computational mesh

Climate Prediction Coprocessor: Instruction Set Architecture (overview)

BIT #	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
Instruction																				
NOP	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	C
NOP	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	C
RESET	0	0	0	0	1	0	0	M	M	M	M	M	M	M	M	M	0	C	C	C
RESET	0	0	0	0	0	1	0	D1	D1	D1	D1	D1	0	0	0	0	0	C	C	C
RESET	0	0	0	1	0	0	1	A1	A1	A1	0	0	0	0	0	0	0	C	C	C
CALL	0	0	1	0	#	#	#	M	M	M	M	M	M	M	M	C	C	C	C	C
JUMP	0	0	1	1	#	#	#	M	M	M	M	M	M	M	M	M	M	C	C	C
MATH	0	1	0	X	#	#	#	S1	S1	S1	S1	S1	S2	S2	S2	S2	D1	D1	D1	C
MATH (@)	0	1	0	X	#	#	#	S	S	S	S	S	S	0	0	0	0	0	0	C
LOGIC	0	1	1	X	#	#	#	S1	S1	S1	S1	S1	S2	S2	S2	S2	D1	D1	D1	C
LOGIC (@)	0	1	1	X	#	#	#	S1	S1	S1	S1	S1	0	0	0	0	0	0	0	C
FLOW	1	0	1	X	#	#	S1	S1	S1	S1	S1	S2	S2	S2	S2	S2	M	M	M	C
DSP(%)	0	1	1	X	#	#	#	S	S	S	S	S	S	M	M	M	M	M	M	C
DSP(%)	0	1	1	X	#	#	#	M	M	M	M	M	M	D	D	D	D	D	D	C

Climate Prediction Coprocessor: Instruction Set Architecture (overview)

<p>@ Where source and destination registers are the same.</p> <p># Instructions from the instruction set.</p> <p>% provisional instruction format</p> <p>M Absolute Address Bit (local memory only).</p> <p>{S Source Register, D Destination Register}</p> <p>C or Condition; varies depending on instruction as to its meaning</p> <p>Privileged instruction mode, X = No privileged instructions in this mode</p> <p>Twenty bit wide instructions allow for about</p> <ul style="list-style-type: none">13100 instructions to be fit into 32k26210 instructions to be held in 64k52400 instructions to be held in 128k	<p>CALL: Jump to memory address, either subroutine or blind.</p> <p>MATH: Mathematical instruction operations: non-Boolean, in floating and Int registers.</p> <p>LOGIC: Boolean operations performed in signed Int registers only.</p> <p>NOP: If condition code CCCCCC=000000, then 1 NOP state, else increment by CCCCCC; this allows up to 64 wait sates to be issued in one instruction.</p> <p>RESET: Resets registers to (+0 or +0.0), CCC = 000 if applicable to register indicated -- else repeat operation for register +1 (CCC = 001) to +7 (111).</p> <p>“NX bit” call should be supported. NX stands for No eXecute. NX technology is used in GPUs to segregate areas of memory for use by either storage of processor instructions (or code) or for storage of data. It is an important CPU security feature to prevent execution of data.</p>
---	--



Climate Prediction Coprocessor: Architecture (register sizes)

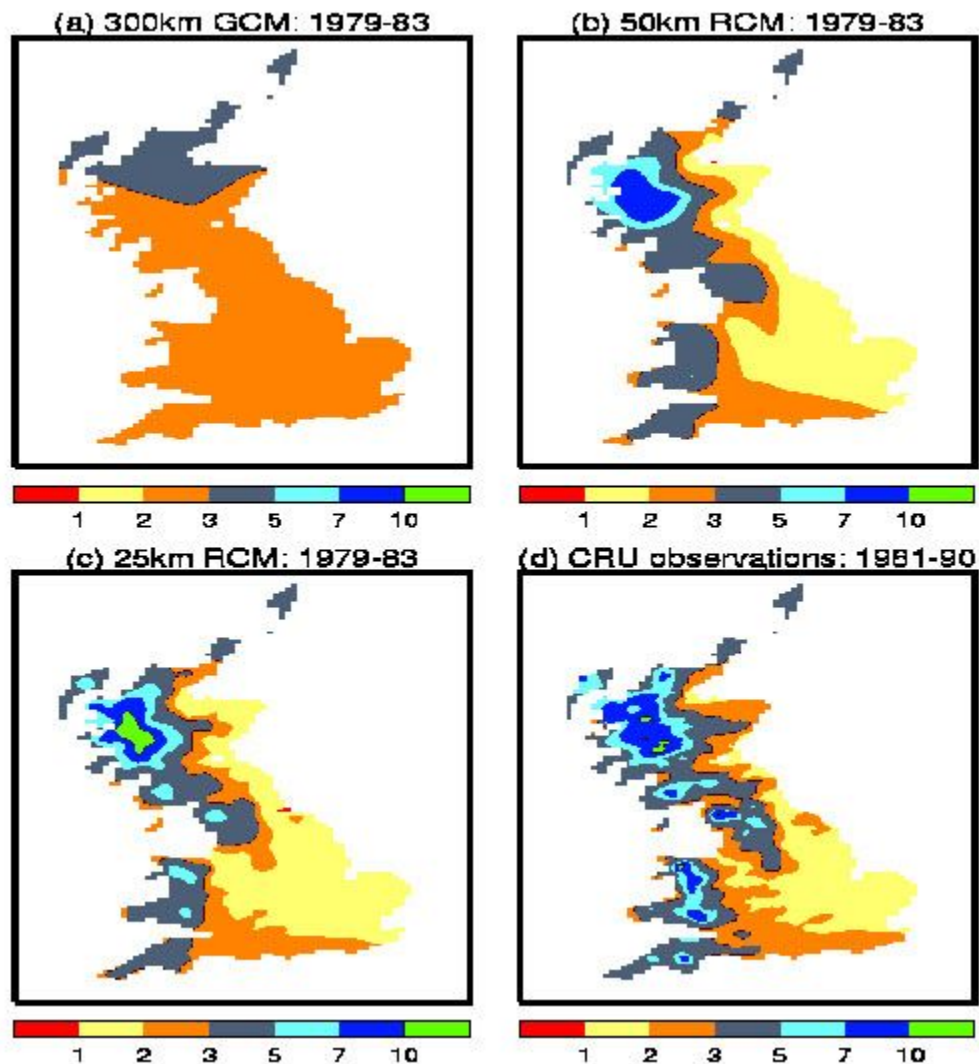
Register type	Register width	Register format	Notes
Floating point	40	F0 to F31	F0.1, 1 st byte; F0.2, 2 nd byte...
Signed Int	40	S0 to S31	S0.1, 1 st byte; S0.2, 2 nd byte...
Address registers	24	A0 to A7	For local address math
Code / Data Demarcation	24	AA	Allow 1k to 63k for program or data
Exception register	40	EE	REG (8 bits) + ERR (4 bits)
CPUID, CPU#	5+32	IDC, IDR	5 Bit ID Call, 32 bit ID Return
MONITOR	5+32	MMC, MMR	5 Bit Call, 32 bit Telemetry Return

How should the Power PC view the individual coprocessors (via the Coprocessor Interface Unit)?

General Principles

These primary design goals should be absolute

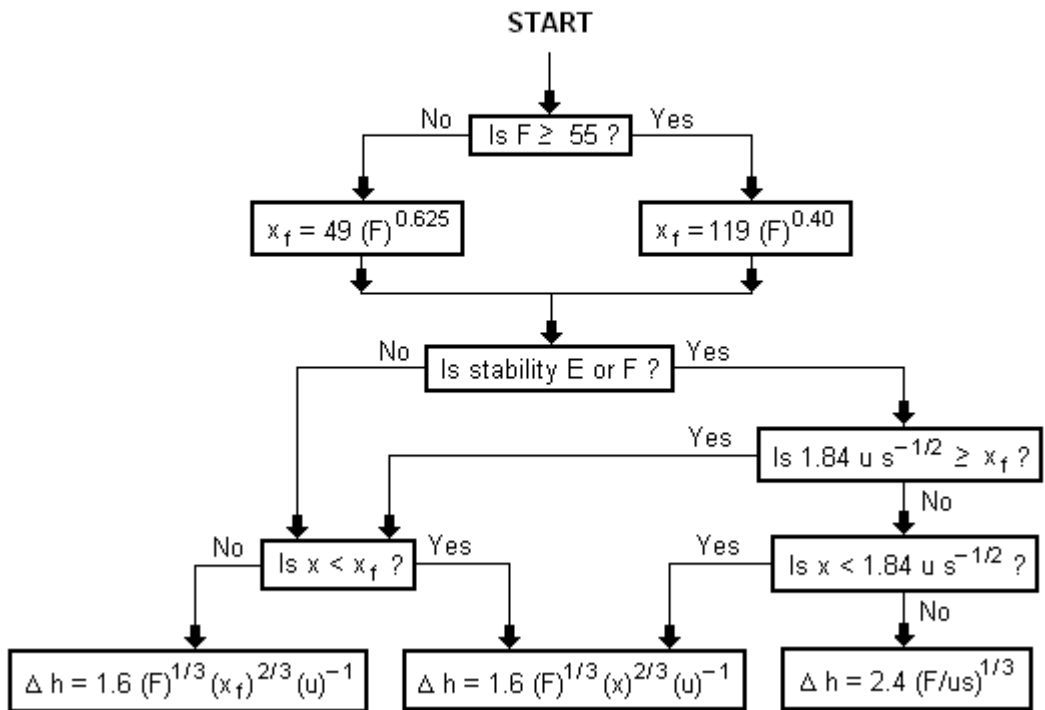
1. Each coprocessor should not be able to write to other coprocessors data or memory spaces.
2. Each coprocessor should only be able to read select memory addresses used by other assigned coprocessors
3. Each coprocessor should be able to “publish” the results of its assigned computation, that is to say the advance of one time step
4. Each coprocessor should only be able to see 2 neighbors or at maximum 6 neighbors, to extract select properties at select addresses



The ideal PC for Numerical Climate Prediction (based on current conditions, coprocessor optional)

- MINIX 3 or QNX operating system, for low operating system overhead
- 6 GB of RAM, 3 GB of it a Virtual RAM Disk
- 8 GB Flash Drive Memory (working data drive, written to once every 15 minutes)
- 1 GB Flash Drive (boot drive)
- 1 x 4 Core CPU, but Power PC “Cell Processor” is preferable or at least viable
- 1 Coprocessor card, like the one described in the text thus far
- 1 x 20 GB Hard Disk, that only powers up every 4 hours for data backup synchronization
- 1 x Ethernet Port
- 1 x WiFi subsystem for wireless networking
- Video Card for setup and monitoring
- 4 USB Ports

LOGIC DIAGRAM FOR BRIGGS' EQUATIONS TO CALCULATE
THE RISE OF A BUOYANT PLUME



Help needed

This CPU and computational research is being done without any funding.

It would be nice to be able to set up a website [with all of the necessary contribution tools] as well as the necessary adjutant infrastructure with an exiting university.

This project should be completely open source and open contribution as possible.

A “Climate Prediction Computational Research Trust” needs to be set up to allow for the long term development of this technology.

Currently, the project needs at least 400,000 SDRs to become viable.

Contact me on bank transfer information if you wish to donate money to this research project.

Climate Prediction (as well as fluid dynamics of the outer planets and the earth's magnetic field) will remain an insurmountable computational problem for many more decades unless a CPU is designed that can cope with the task with as little added complexity to the host system as possible.

The coprocessor should always be able to used to conduct cryptographic research, and other low complexity non-vortex research – as being too application specific will result in silicon being wasted.

<u>Created by</u>	<u>Original Concept</u>	<u>Last Modified</u>	<u>Last Revision</u>	<u>Revision State</u>
Max Power	11/22/08	02/27/14	See Tracking	0.55r928

Revision Tracking

Date	Changes Made
02/25/14	Add 2 GPU types, delete Random instruction, fix readability
02/27/14	Add Flow Control Instructions and Math Constants