Efficient and accessible? Addressing new architectures in C++

Robin Williams

April 20, 2016

Outline

Part I

The Challenge

The Challenge

- Why do we need more compute?
- Hardware trends
- Object oriented code

Why do we need more compute?

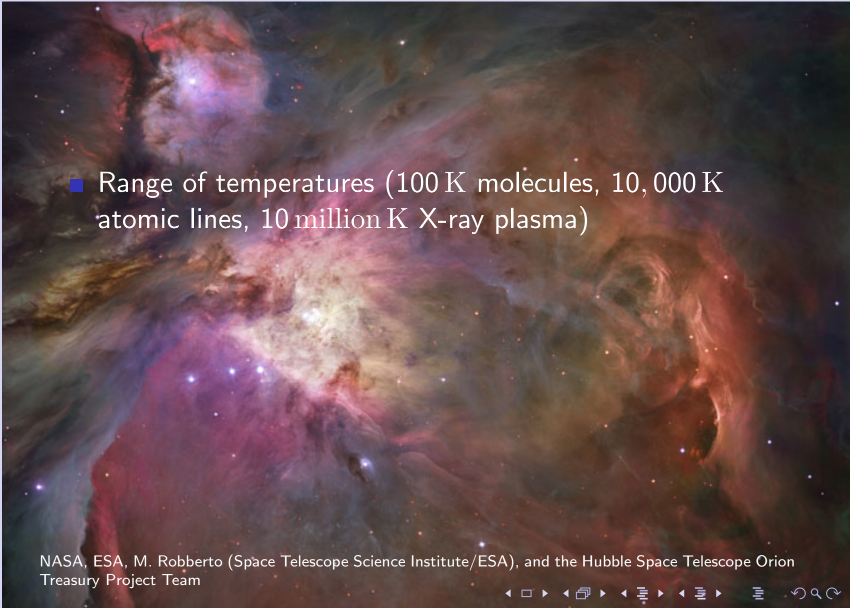
- Virtual Realities
  - Complex system modelling
  - Games
  - Prettier GUIs!
- Big data
  - Google searches – $3\,\mathrm{billion}/\mathrm{day}$
  - Twitter tweets – $70\,\mathrm{Gb}/\mathrm{day}$
  - LHC – $25\,\mathrm{PB}/\mathrm{yr}$
  - SKA – $300\,\mathrm{PB}/\mathrm{yr}$
  - Human genome – $\sim 1.6\,\mathrm{GB}$, $200\,\mathrm{GB}$ sequencer data
  - 1024^3 fluid dynamics data $\sim 40\,\mathrm{GB}$

"In view of its rapidity of action, and of the ease with which it can be switched over from one type of problem to another it is very possible that the one machine would suffice to solve all the problems that are demanded of it from the whole country."

Sir Charles Darwin, NPL, 1946

NASA, ESA, M. Robberto (Space Telescope Science Institute/ESA), and the Hubble Space Telescope Orion Treasury Project Team

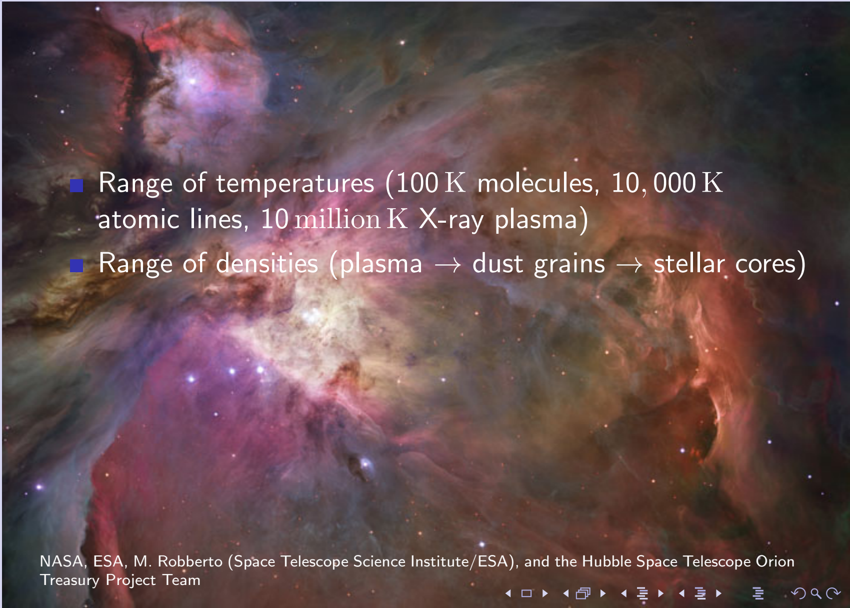
A Complex System: The Orion Nebula



- Range of temperatures (100 K molecules, 10,000 K atomic lines, 10 million K X-ray plasma)

NASA, ESA, M. Robberto (Space Telescope Science Institute/ESA), and the Hubble Space Telescope Orion Treasury Project Team

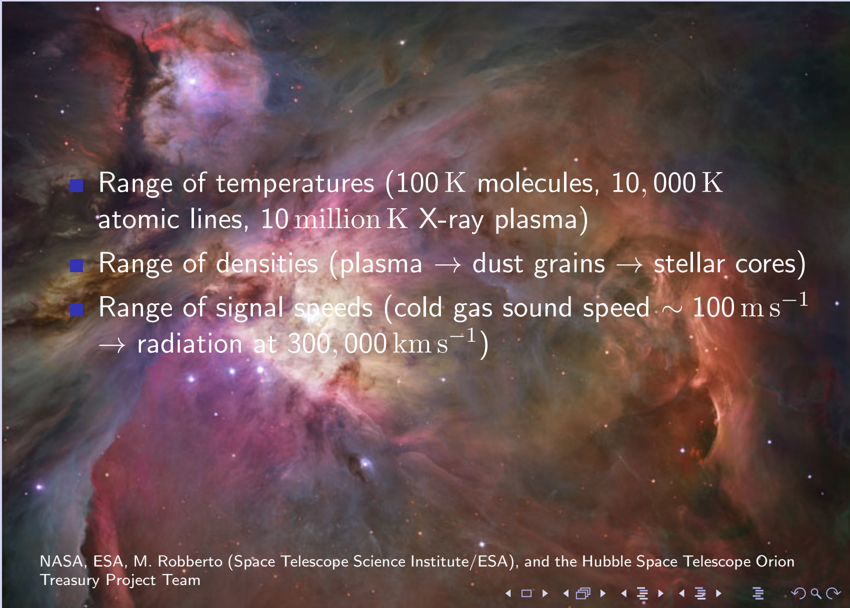
A Complex System: The Orion Nebula



- Range of temperatures (100 K molecules, 10, 000 K atomic lines, 10 million K X-ray plasma)
- Range of densities (plasma → dust grains → stellar cores)

NASA, ESA, M. Robberto (Space Telescope Science Institute/ESA), and the Hubble Space Telescope Orion Treasury Project Team

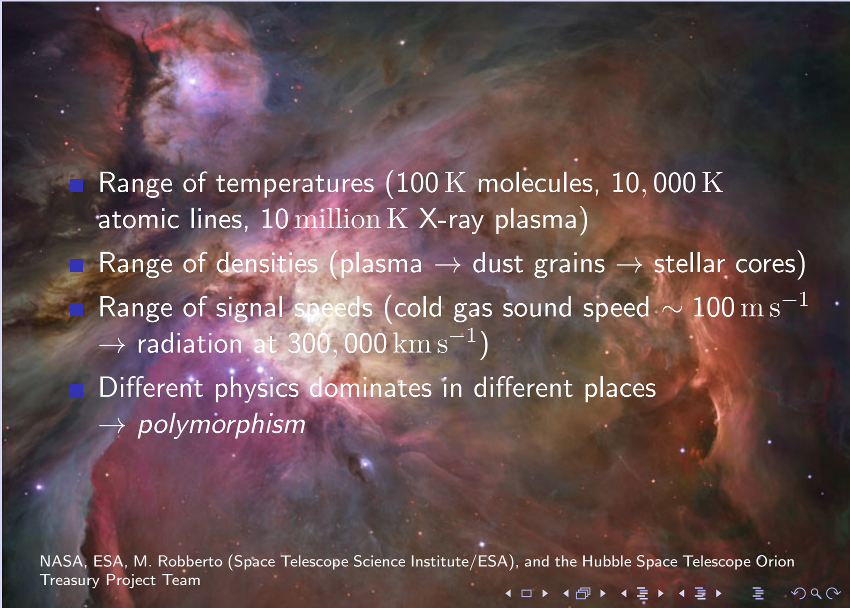
A Complex System: The Orion Nebula



- Range of temperatures (100 K molecules, 10, 000 K atomic lines, 10 million K X-ray plasma)
- Range of densities (plasma → dust grains → stellar cores)
- Range of signal speeds (cold gas sound speed $\sim 100\,\mathrm{m\,s^{-1}}$ → radiation at $300,000\,\mathrm{km\,s^{-1}}$)

NASA, ESA, M. Robberto (Space Telescope Science Institute/ESA), and the Hubble Space Telescope Orion Treasury Project Team

A Complex System: The Orion Nebula



- Range of temperatures (100 K molecules, $10,000\,\mathrm{K}$ atomic lines, $10\,\mathrm{million}\,\mathrm{K}$ X-ray plasma)
- Range of densities (plasma $\rightarrow$ dust grains $\rightarrow$ stellar cores)
- Range of signal speeds (cold gas sound speed $\sim 100\,\mathrm{m\,s^{-1}}$ $\rightarrow$ radiation at $300,000\,\mathrm{km\,s^{-1}}$)
- Different physics dominates in different places $\rightarrow$ *polymorphism*

NASA, ESA, M. Robberto (Space Telescope Science Institute/ESA), and the Hubble Space Telescope Orion Treasury Project Team

Personal background

- I'm interested in modelling emission nebulae
  - Finding equilibrium models to predict their spectra
  - Modelling their development in time
- Data access patterns are different from web/account serving
  - Tend to loop through all the working set once per timestep
- But not so different from data mining?

Hardware trends

- Moore's Law – finer lithography, more transistors
  - Super-scalar dispatch
  - More cores
  - More vectorization
- Heterogeneous architectures: APUs, (GP)GPUs, MIC
- Clock rate saturation – capacitative bucket equation

  $$P \simeq \alpha C V^2 f$$

  where capacitance is $C \simeq \epsilon A/d$
- Deep cache hierarchies
  - Small, fast, low-latency on-die memory

Dual core revolution, 150 million BC

Dual core revolution, 150 million BC

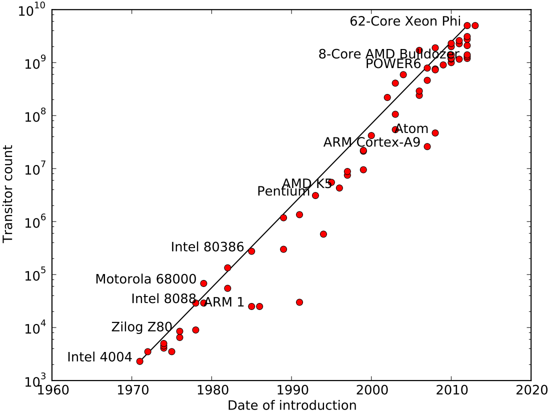
Francisco Rollandin, openclipart

Dual core revolution, 150 million BC

Francisco Rollandin, openclipart

Sadly, it's a myth...

Moore's law

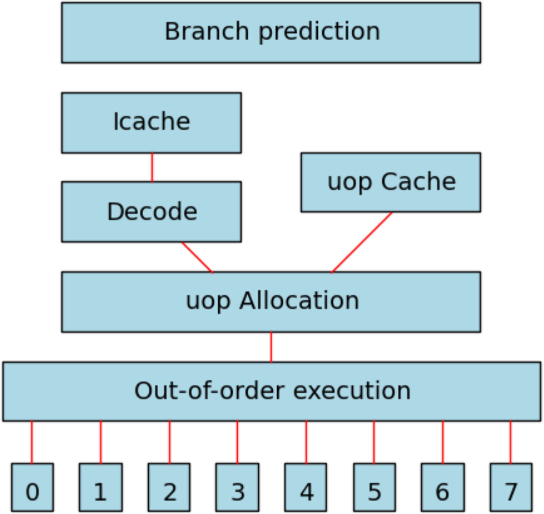


Data source: Wikipedia

Some hardware

	6502	Cray 2	i7 3770	Human Brain
Year	1975	1985	2012	
System mass	$\sim$ 4 kg	2500 kg	$\sim$ 10 kg	$\sim$ 70 kg
Processor mass	$\sim$ 10 g		$\sim$ 200 g	$\sim$ 1.5 kg
Transistor count	3510		1.4×10^9	1.2×10^{11}
Half-pitch	8 μm	$\sim$ 1 μm	22 nm	4–100 μm
Feed-in	few	few	few	10^3–10^4
TDP	0.7 W	200 kW	77 W	20 W
Compute cores	1	$1 + 4$	$4 + 16$	
Data registers	1	$8 + 8 \times 64v$	$16 + 16 \times 8v$	5 ± 2
Speed	0.43 MIPS	1.9 GFlops (64b)	112 + 73.6 GFlops (64b)	10^{13-16} op s^{-1}
Clock speed	1 MHz	250 MHz	3.4–3.9 GHz	$\sim$ 10 Hz
Memory clock	1 MHz	250 MHz	200 MHz	
Main memory	64 kiB	2 GB	$\rightarrow$ 32 GB	2.5 PB
Bandwidth	$\sim$ 1 MB/s	$\sim$ 4 GB/s	25.6GB/s	
All-to-all rate	15 Hz	2 Hz	0.8 Hz	3 Hz

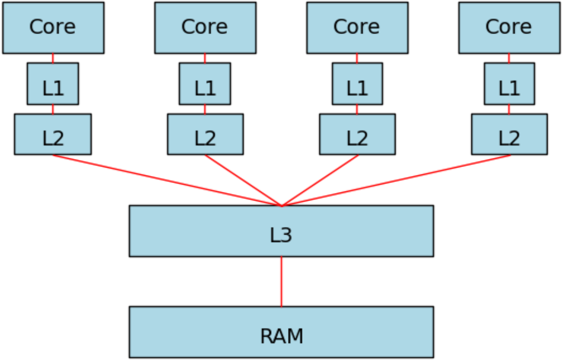
Instruction pipeline



Ivy Bridge code pipeline within core

- CISC instructions decoded to RISC μops
  - Can read & decode 4 instructions per cycle
- Decoded μops passed to reorder buffer
  - taken from queue out-of-order, based on dependencies
- 6 execution ports operate simultaneously, e.g. for floating
  - 0: mul (256b, $\ell = 5$), div & sqrt (128b, $\ell = 10$–22)
  - 1: add (256b, $\ell = 3$)
  - 2&3: Memory read, 128b
  - 4: Memory write, 128b (using address from 2,3)
  - 5: mov, shuffle, boolean (256b, $\ell = 1$)
- High performance requires
  - operation balance (mul/add in parallel)
  - streaming through execution pipelines

Logical memory architecture

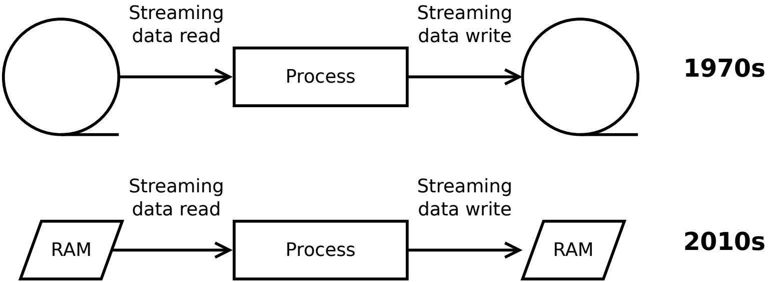


Memory hierarchy, e.g. Ivy Bridge

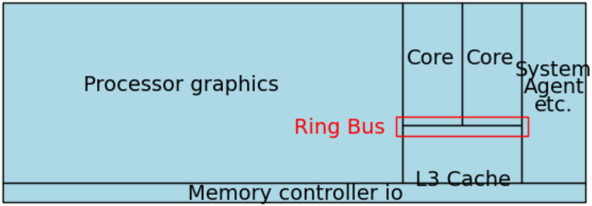
Cache	Size	Latency	Bandwidth[1]
L1D	32kB	4–5 cycles	320 GB/s
L1I	32kB	4–5 cycles	
L2	256kB	12 cycles	173 GB/s
L3 (shared)	8Mb	29.5/30.5 cycles	103 GB/s
RAM (shared)	~8Gb	30 cycles + 53 ns	20.5 GB/s

[1]Source: `http://www.sisoftware.net/?d=qa&f=cpu_ivb`

Back To The Future



Physical CPU architecture



Intel Broadwell processor die, to scale (roughly)
Dual core, HD6000/Iris 6100 graphics: $133\,\mathrm{mm}^2$

An object-oriented code fragment

```
for (auto tr=trans.begin(); tr != trans.end(); ++tr)
{
  state *lo = tr->lo;
  state *hi = tr->hi;
  tr->popopc = lo->pop - hi->pop*lo->g/hi->g;
}
```

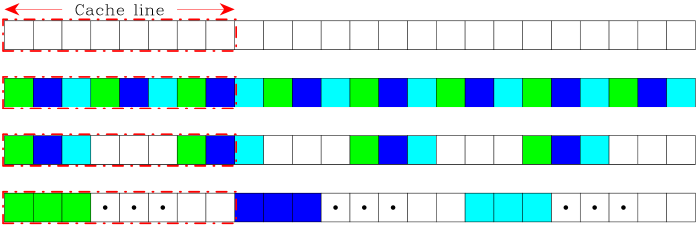
Data structure


```
class transition {
public:
  float popopc;
  state *lo, *hi;
  // ...and space for other object properties (LOTS)
  float padding[NPAD];
};
```

Timings

Vector size	NPAD=0	NPAD=12
100	3.373	4.110
200	3.550	5.673
500	3.579	5.719
1000	3.571	5.767
2000	3.573	5.834
5000	3.584	10.682
10000	3.846	12.274
20000	4.443	12.651
50000	4.670	12.844
100000	4.774	12.853

Container memory usage



- Cache line 64b (i.e. 16 float, 8 double)
- More class members ⇒ inefficient cache usage, overlaps
- Structure-of-arrays allows streaming reads, alignment; keeps (currently) irrelevant data out of view
- Contention possible, but 8-way associative caches now typical

Try moving data into multiple vectors

```
int ntrans = popopc.size();
for (int nt=0; nt<ntrans; ++nt)
{
  int lo = los[nt];
  int hi = his[nt];
  popopc[nt] = statepop[lo] - statepop[hi]*stateg[lo]
                                          /stateg[hi];
}
```

Timings

Vector size	NPAD=0	NPAD=12	Vector
100	3.373	4.110	3.526
200	3.550	5.673	3.540
500	3.579	5.719	3.572
1000	3.571	5.767	3.571
2000	3.573	5.834	3.570
5000	3.584	10.682	3.579
10000	3.846	12.274	3.580
20000	4.443	12.651	3.594
50000	4.670	12.844	3.633
100000	4.774	12.853	3.630

Vector and pre-division


```
float *staterat = new float[nstate];
for (int i=0; i<nstate; ++i)
{
  staterat[i] = statepop[i]/stateg[i];
}
for (int nt=0; nt<ntrans; ++nt)
{
  int lo = los[nt];
  int hi = his[nt];
  popopc[nt] = statepop[lo]-staterat[hi]*stateg[lo];
}
delete [] staterat;
```

Timings

Vector size	NPAD=0	NPAD=12	Vector	Vec&Pre-div
100	3.373	4.110	3.526	1.647
200	3.550	5.673	3.540	1.655
500	3.579	5.719	3.572	1.672
1000	3.571	5.767	3.571	1.695
2000	3.573	5.834	3.570	1.685
5000	3.584	10.682	3.579	1.703
10000	3.846	12.274	3.580	1.704
20000	4.443	12.651	3.594	1.748
50000	4.670	12.844	3.633	1.835
100000	4.774	12.853	3.630	1.837

Conclusions

- Other options possible, breaking up operations
- Results vary in detail between systems
- Putting data into vectors is better, even without padding
- Resulting code is very 'low-level'
- Difficult to measure cache-sensitive operations
  - `CLFLUSH` operations?

Timings – SSE/AVX

Vector size	NPAD=0	V&P std	V&P -march=native	Optimized arch=native
100	3.373	1.647	2.103	1.528
200	3.550	1.655	2.042	1.509
500	3.579	1.672	2.059	1.584
1000	3.571	1.695	2.057	1.601
2000	3.573	1.685	2.057	1.594
5000	3.584	1.703	2.071	1.626
10000	3.846	1.704	2.081	1.690
20000	4.443	1.748	2.109	1.922
50000	4.670	1.835	2.162	2.080
100000	4.774	1.837	2.176	2.115

- Exploiting SSE/AVX needs *fewer* levels of indirection

OpenMP


```
#pragma omp parallel for
  for (auto tr=trans.begin(); tr != trans.end(); ++tr)
  {
    state *lo = tr->lo;
    state *hi = tr->hi;
    tr->popopc = lo->pop - hi->pop*lo->g/hi->g;
  }
```

OpenMP


```
#pragma omp parallel for
  for (auto tr=trans.begin(); tr != trans.end(); ++tr)
  {
    state *lo = tr->lo;
    state *hi = tr->hi;
    tr->popopc = lo->pop - hi->pop*lo->g/hi->g;
  }
$ make
speedtest.cpp: error: invalid controlling predicate
  for (auto tr=trans.begin(); tr != trans.end(); ++tr)
```

Timings – OpenMP

Vector size	NPAD=0	V&P	OMP×4
100	3.373	1.647	1.300
200	3.550	1.655	0.897
500	3.579	1.672	0.615
1000	3.571	1.695	0.514
2000	3.573	1.685	0.474
5000	3.584	1.703	0.445
10000	3.846	1.704	0.439
20000	4.443	1.748	0.808
50000	4.670	1.835	1.503
100000	4.774	1.837	1.521

Vector-friendly code

```
for (int nt=0; nt<ntrans; ++nt)
{
  int lo = los[nt];
  int hi = his[nt];
  popopc[nt] = statepop[lo]-staterat[hi]*stateg[lo];
}
```

- This looks very F77 – is a halfway house possible?

Structure-of-Array 'containers'


```
struct ObjectList
{
  std::vector<float> m_a, m_b;
};
struct ObjectIterator
{
  ObjectList *m_list;
  int m_index;
};
```

- Iterator contains pointer to list, and index
- `operator++` etc. are fairly obvious
- But how do we provide access to `a`, `b`, etc.?

Proxy Objects


```
const ObjectProxy& ObjectIterator::operator*();
const ObjectProxy* ObjectIterator::operator->();
```

- Use a proxy object to provide access
- Reference-like semantics
- Rules for overloading `operator->` pose a challenge

  If used, its return type must be a pointer or object of a class to which you can apply ->.

  Stroustrup, C++ PL, 3rd ed.

- The second part goes around in a self-referential loop
- What (real!) pointer can we provide?

A proxy object with a multiple personality

- The hint is the data the pointed-to object must hold
  - `ObjectList *m_list;`
  - `int m_index;`
- Just like the iterator!!
- Can we use the iterator itself as the pointed to object?
  - We know it exists
  - We know it continues to exist
  - We know it has the right content
- Need to segregate the data-access from the iterator-fu
  - Use the same object, *just with a different static type*

Multiple personality implementation

- Several implementation options
  - Private inheritance? LSP $\Rightarrow$ prefer containment

```
class ObjectProxy
{
  friend class ObjectIterator;
  ObjectList *m_list;
  int m_index;
};
class ObjectIterator // Can be a template
{
  ObjectProxy p;
public:
  ObjectProxy* operator->() { return &p };
};
```

Some problems remain...

```
class ObjectProxy
{
public:
  float a() const { return m_list->a[m_index]; }
  float& a() { return m_list->a[m_index]; }
};
```

- Data access requires ugly ()
  - Attributes? (Fiddling with type conversions?)
- Adding a component requires multiple touch-points
  - List definition, resize, proxy accessors & const-accessors, copy
- Some STL `sorts` obstruct reference-*like* semantics
- Polymorphism (a problem for standard containers)

Polymorphic containers

- Traditional approach – C++
  - `std::vector<std::unique_ptr<Base> >`
  - Another level of indirection
  - Entire contents of class now (potentially) scattered
  - Very poor cache locality, heap fragmentation
  - Additional memory for pointers
- Traditional approach – Fortran
  - Add indexes to separate lists of extra components
    - Requires indirect access for these
    - Probably in wrong order
    - Index elements use memory for *all* cells

Improving the Fortran approach

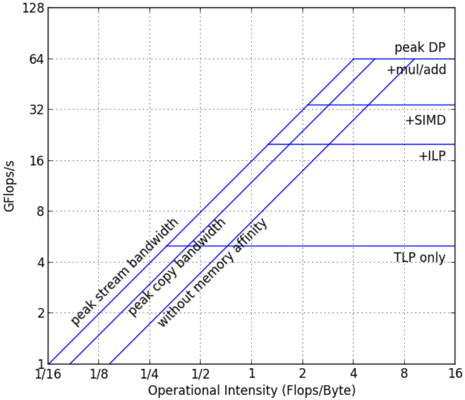
- De-fragment indirect list
  - Improves cache locality for mixed cell components
  - Can also apply to local AMR (see below)
- De-fragmentation groups similar cells
  - ‘Business’ problems object ordering often arbitrary
  - Dominant physics often spatially clustered, e.g. molecular clouds
- Run-length encode index array
  - Reduces size
  - Enables vectorization within each RLE’d section

Measurement and analysis

> *Measurement is a crucial component of performance improvement since reasoning and intuition are fallible guides...*
>
> Kernighan & Pike 1999, The Practice of Programming

- Measurement, e.g.
  - Perf
    - Fast, accesses CPU performance registers, can multiplex multiple events
  - Likwid
    - Can trace evolution of load through time
  - Cachegrind
    - Slow but deterministic
- Analysis
  - Roofline model

Roofline model



- Plot not to scale
- Williams, Waterman & Patterson (1999), Comm ACM.

Throughput-limited loop

```
/* increment the fine opacity array */
for( long i=0; i<rfield.nfine; ++i )
{
  realnum tauzone = rfield.fine_opac_zone[i]*
    radius.drad_x_fillfac;
  rfield.fine_opt_depth[i] += tauzone;
}
```

- `rfield.nfine` is big...
- Can we do something else with the data while its in-core?

Part II

Tools

Some available technologies

- Compiler optimization
- Cache optimization of native code
- Vector intrinsics
- OpenMP/OpenACC
- OpenCL
- C++AMP
- MPI

Compiler optimization

- Easy
  - So long as your code meets requirements...
  - May get better just by waiting (e.g. gcc vs clang vectorization shoot-out)
- Obtuse
  - Outcome isn't transparent and may not be portable
  - ...if the compiler doesn't know the opcode, it can't use it
- Features may be default for `-O3`, or require flags or pragmas
- Compile-time only
  - But can use profile-guided optimizations
- Hand-tuning may not be robust, e.g.
  - `__restrict__` makes little difference now
  - Explicit pre-fetching can be worse than default

Cache optimization of native code


```
for (int j=0; j<N; ++j) {
  for (int i=0; i<N; ++i) {
    a[i][j] = b[i][j] + c[i][j];
  }
}
```

- Simplest example of cache sub-optimization
- Substantial performance advantage by swapping loops
- How about Fortran?

Fortran 90 cache optimization

```
a = b + c
```

- Minimal code, and just does the right thing!
- *Might* also work in C++, depending on array class...

N-body code force loop


```
for (int np1=0; np1<NP; ++np1) {
  for (int np2=0; np2<np1; ++np2) {
    f = force(x[np1],x[np2]);
    fp[np1] += f; fp[np2] -= f;
  }
}
```

- Inner loop cycles through memory $\sim$ NP times

Cache blocking


```
for (int np2a=0; np2a<NP; ++np2a += NB) {
  for (int np1=0; np1<NP; ++np1) {
    for (int np2=np2a;np2<np1&&np2<np2a+NB;++np2) {
      f = force(x[np1],x[np2]);
      fp[np1] += f; fp[np2] -= f;
    }
  }
}
```

- If `NB` matches cache block, memory traffic $\Downarrow$ by `NB`
- Innermost loop can be unrolled (with care)
- `NB` requires tuning, code more opaque
  - Intel's OpenMP matrix-multiply example: 7 $\rightarrow$ 34 loc, with double-parking

OpenMP

- Well-established & supported technology, since 1997
- Applied to shared-memory systems – low barrier to entry
- `#pragma` directives apply to succeeding loop
- Rather opaque relation to the underlying hardware
- OpenACC extends the pragma approach to accelerators
  - Recent ($\sim$ 2011) development, limited support to date

Vector intrinsics

- Supported by gcc, Intel, Microsoft compilers
- Availability varies with hardware
- Low level – which may impede other optimizations
- Bound to non-standard data types

```
short int bb[]={1,2,3,4,5,6,7,8},
  cc[]={8,7,6,5,4,3,2,1},aa[8];
__m128i b=LoadVector(bb), c=LoadVector(bb);
// Multiply b and c
__m128i bc = _mm_mullo_epi16 (b, c);
StoreVector(aa,bc);
```

- Ends up looking like assembler...

OpenCL

- Recent (since 2008), intended to converge vendor-specific GPGPU frameworks
- Internal C-like language to execute on multiple devices
- Restrictions
  - No pointers to functions
  - No pointers-to-pointers in kernel arguments
  - No bit-fields, variable length arrays or structures
  - No recursion
  - Double types are an optional extra
- Kernels sent to device, and executed asynchronously

The BIG idea behind OpenCL – replace loops with functions executing at each point in a problem domain

McIntosh-Smith & Deakin

OpenCL v1.2 Memory Model

- Host Memory
- Device Memory
  - Global Memory – RW for all work-items in all work-groups
  - Constant Memory – Read only, constant for kernel
  - Local Memory – Private to work-group
  - Private Memory – Private to work-item
- Global & Constant Memory can be cached
- Global, Local & Private Pointers may be cast to – but not from – a single *generic address space*
- API commands provided to copy data to/from Global & Constant Memory

OpenCL evolution

- OpenCL 1.2 $\Rightarrow$ 2.0
  - Shared virtual memory – fine-grained access to host memory space
  - Device queues
  - Pipes
- Moving towards a actor/dataflow model
- Full SVM model requires hardware support for efficient implementation
  - Platform-dependent coding may be required

C++AMP

- Extension to the C++ language, initially proposed by Microsoft
- Targeted at heterogeneous architectures
- Defines restriction operators for data and functions, to document code portability level
- Philosophy seems similar to C/C++ headers
  - Advertise a contract
  - Test against it
  - But C++ may move to modules...

MPI

- Distributed memory parallelization framework
- Well supported & widely available
- Long history in HPC – was big parallel before parallel was big
- Actually pretty flexible
- Often used in a lock-step manner
  - efficient on old, 'clean' architectures
  - can exacerbate resource contention

Framework sizes

- OpenMP v4.0 (July 2013) API specification: 312pp
- OpenACC v2.0a (August 2013) API specification: 74pp
- OpenCL v2.0 (July 2015) API specification and language: 288+202pp
- C++AMP v1.2 (December 2013) Language and Programming Model: 184pp
- Intel(R) C++ Intrinsic Reference: 193pp (to SSE4, SIMD from p24 on)
- MPI v3.1 (June 2015) API specification: 868pp

Part III

Transcendence

Back To The Future – Part II

Before C, there was far more hardware diversity than we see in the industry today. Computers proudly sported not just deliciously different and offbeat instruction sets, but varied wildly in almost everything, right down to even things as fundamental as character bit widths (8 bits per byte doesn't suit you? how about 9? or 7? or how about sometimes 6 and sometimes 12?) and memory addressing (don't like 16-bit pointers? how about 18-bit pointers, and oh by the way those aren't pointers to bytes, they're pointers to words?).

http://herbsutter.com/2011/10/12/dennis-ritchie/

Personal Computer World WALLCHART

BASIC CONVERTER CHART

Personal Computer World WALLCHART

BASIC RESERVED WORDS & FORMATS

Portability

One day, all computers will understand the same language (and read each others' disks and address the screen in the same way and . . .). To tide you through until this great day arrives, however, we set out to beg, steal or even buy thirteen of the most popular home micros to produce this fully revised, corrected and updated PCW Basic Converter Chart.

Whether you're trying to convert that amazing Spectrum game to run on your Oric, have just spent the past three hours wondering why your new BBC micro doesn't seem to support a FRE statement or simply want to write programs which can be easily converted to other micros, the PCW Basic Converter Chart is here to help.

Take a step backwards

With caches, contention & speculation, every detail is fragile

Strategies

- My first threaded program (circa 1998)
- PDL
- Aqualung
- AMD Bolt/Khronos SyCL
- pycuda.gpuarray.GPUArray
- Parallel made *really* easy
- Fundamental algorithm
- Expression templates
- Software/hardware ecosystem

My first threaded program (circa 1998)

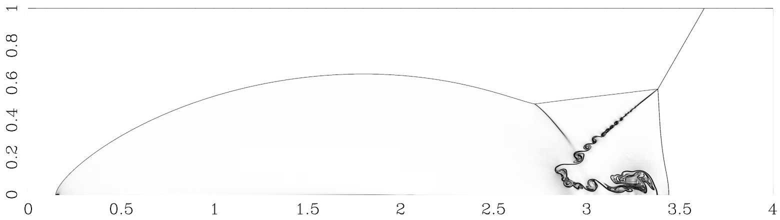

```
use PDL; use PDL::Graphics::TriD;
$b=zeroes(50,50,50);
$b=sin(0.3*$b->rvals)*cos(0.3*$b->xvals);
$c=0; $a=byte($b>$c);
foreach(1,2,4) {
  $t = ($a->slice("0:-2")<<$_);
  $t += $a->slice("1:-1");
  $a = $t->mv(0,2);
}
keeptwiddling3d();
points3d [whichND(($a != 0)&($a != 255))];
```

What does this show?


```
use PDL;use PDL::Graphics::TriD;keeptwiddling3d();$b=zeroes(50,50,
50);$b=sin(0.3*$b->rvals)*cos(0.3*$b->xvals);$c=0;$a=byte($b>$c);
foreach(1,2,4){$t=($a->slice("0:-2")<<$_);$t+=$a->slice("1:-1");
$a=$t->mv(0,2);} points3d [whichND(($a != 0) & ($a != 255))];
```

- Do what...?
  - It's a perlsig ;-)
  - It's (nearly) surface rendering!
- OK, but where is the threading?
  - It's there because *there's nothing to say it isn't!*
  - Using minimal loops expresses algorithm directly
  - Built into the core of PDL by Tuomas J. Lukka

Aqualung



- Adaptive mesh refinement fluid dynamics code

  `http://adsabs.harvard.edu/abs/2000MNRAS.316..803W`
- Collela-Woodward test problem: a shock hits a wedge
  - Up to 1,200,000 AMR cells, cf 67,108,864 at full resolution

Aqualung Mesh

Efficient and accessible?

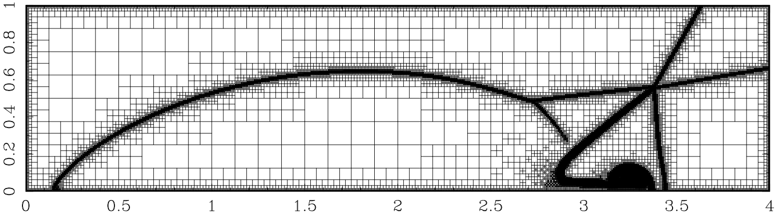
Robin Williams

The Challenge

Tools

Transcendence

Conclusions



- Quad- (more generally oct-) tree local refinement
- A Unix system programmer's approach to AMR...

Aqualung parallelization

- Numerical method expressed as operator mappings over mesh
  - Compare the STL iterator/algorithm split, or MapReduce
- Retro-fitted parallelization
- OpenMP parallelized by placing a single directive in iterator template
- Later switched to pthreads task pool without changing main code
- Limited scaling emphasizes importance of memory management

Aqualung example


```
// ========== iterate.h ==========
template<typename F, typename L>
void iterate(const F &f, const L &l, node_addr n,
  Grid &grid, const Context &ctx) {
#if defined _OPENMP
#pragma omp parallel for
#endif
  for (node_addr i=0; i<n;++i)
    f(l[i],i,grid,ctx);
}
// ========== aqualung.cpp ==========
// prim => iflux -- but beware false sharing
    iterate(Fluxes1(prim,iflux),
      interfaces, interfaces.size(), grid, ctx);
```

AMD Bolt Library

- STL-compatible template library for GPU acceleration
- Can convert serial to GPU code just by changing namespaces

```
std::vector<float> a(n), b(n), r0(n), r1(n);
Functor f;
std::transform(a.begin(), a.end(), b.begin(),
  r0.begin(), f);
bolt::cl::transform(a.begin(), a.end(), b.begin(),
  r1.begin(), f);
```

https://github.com/HSA-Libraries/Bolt

- Similar approach in e.g. Khronos SyCL

pycuda.gpuarray.GPUArray

- Abstracts away CUDA boiler-plate
- Just document moves of data to & from arrays on the GPU
- Script variables act as handles, passing messages to GPU
- Coding can be substrate-agnostic until it needs to move devices

```
a_gpu = gpuarray.to_gpu(numpy.random.randn(4,4).
    astype(numpy.float32))
a_doubled = (2*a_gpu).get()
```

http://documen.tician.de/pycuda/tutorial.html

```
make -j
```

Fundamental algorithm

```
get data from sources
process data
store data
```

- Data comes from multiple sources, movement needs to be managed
- Time to process may not be predictable
  - History probably a better guide than analysis $\Rightarrow$ process can be opaque
- Can the tasks be split or combined?
  - Data dependencies need to be transparent

Expression templates

- Can generate vectorized coding using operator overloading

```
Vec operator+(const Vec& a, const Vec& b);
Vec& operator=(const Vec& a);
    Vec a, b, c;
    a = b + c;
```

- But a=b*c+d generates multiple temporaries
- Instead, can use lazy evaluation, e.g.

```
Vop operator+(const Vec& a, const Vec& b) {
  return Vop2(Vop2::Add, a, b);
}
```

- Blitz++ pioneered this approach (Veldhuizen 1994, also Vandevoorde 1995)

Pros and Cons of Expression Templates

- Execute operations in a lazy manner, without temporaries
- Operator overloading can effectively hijack the C++ parser
  - Resulting object hierarchy replicates the AST
- Execution can be dispatched to multiple agents based on dataflow & capability information
  - Starts to work more like a dynamic language interpreter
  - Could automatically generate OpenCL kernels
- Can't overload conditionals (`if` and `?:`)
  - Convert to `merge()`
  - Vector code typically executes both branches anyway
- With power comes responsibility
  - Have undermined compiler optimizations

Vector second-order advection


```
for (int sub=0; sub<2; ++sub)
{
  float dtsub = dt*float(1+sub)/2;
  vortex::vector<float> ge = rho-rho.rotate(-1);
  vortex::vector<float> ga = choose(ge>0.f,ge,-ge);
  vortex::vector<float> g = choose(ga<ga.rotate(1),
    ge,ge.rotate(1)); // MinMod limiter
  vortex::vector<float> f = rho+u*g;
  rho = rho0+(dtsub/dx)*(f.rotate(-1)-f);
}
```

- Loop-free vector code unfamiliar
  - But numerical algorithm developers often use MatLab

Immutable data

- Data locks can be a major sink of time in parallel code
  - Access to immutable data doesn't need sequencing
- Simple application of immutability creates yet more temporaries $\Rightarrow$
  - More dynamic memory allocation
  - More cache turnover (but overall R+W dataflow similar)
- Maintaining execution context & use counts allows temporaries to be dynamically elided

Turing completeness vs. the machine model

- Very minimal structure needed to be computationally complete
- But languages are dragged towards the machine
  - To use the hardware, need to know the opcodes
  - Simplest to implement a direct mapping
- Conversely, machines can also be dragged towards the language
  - Influence of C machine model on architectures

Requiem for C?

...the report of my death was an exaggeration.

Mark Twain

- Hardware and compiler development have meant for a long while that C-family languages aren't really that 'close to the metal'
- The metal hasn't gone away – it's ripping its way through the fabric of abstractions underlying C-family languages...
- ...causing a proliferation of *new* C-like languages
- One would *hope* that time will redux this to new compact abstractions

So where do we need to be going?

- Optimization of resource utilization must be dynamic
  - Need to deal with cross-talk effects
    - memory channel saturation
    - cache contention
    - interrupts
    - varying core speed with load and ambient temperature...
  - Static optimization can't deal with dynamic conditions

So where do we need to be going?

- Optimization of resource utilization must be dynamic
  - Need to deal with cross-talk effects
    - memory channel saturation
    - cache contention
    - interrupts
    - varying core speed with load and ambient temperature...
  - Static optimization can't deal with dynamic conditions
- How to address this?
  - Can use expression templates to capture algorithm
  - Execute on an dynamically-scheduled vector virtual machine

So where do we need to be going?

- Optimization of resource utilization must be dynamic
  - Need to deal with cross-talk effects
    - memory channel saturation
    - cache contention
    - interrupts
    - varying core speed with load and ambient temperature…
  - Static optimization can't deal with dynamic conditions
- How to address this?
  - Can use expression templates to capture algorithm
  - Execute on an dynamically-scheduled vector virtual machine
- *Déjà vu?*
  - Just like μop dispatch to execution ports, kernel dispatch in GPGPUs

Part IV

Conclusions

Conclusions

- For performance, use `std::vector<>`

Conclusions

- For performance, use `std::vector<>`
- For performance, use `std::vector<native>`

Conclusions

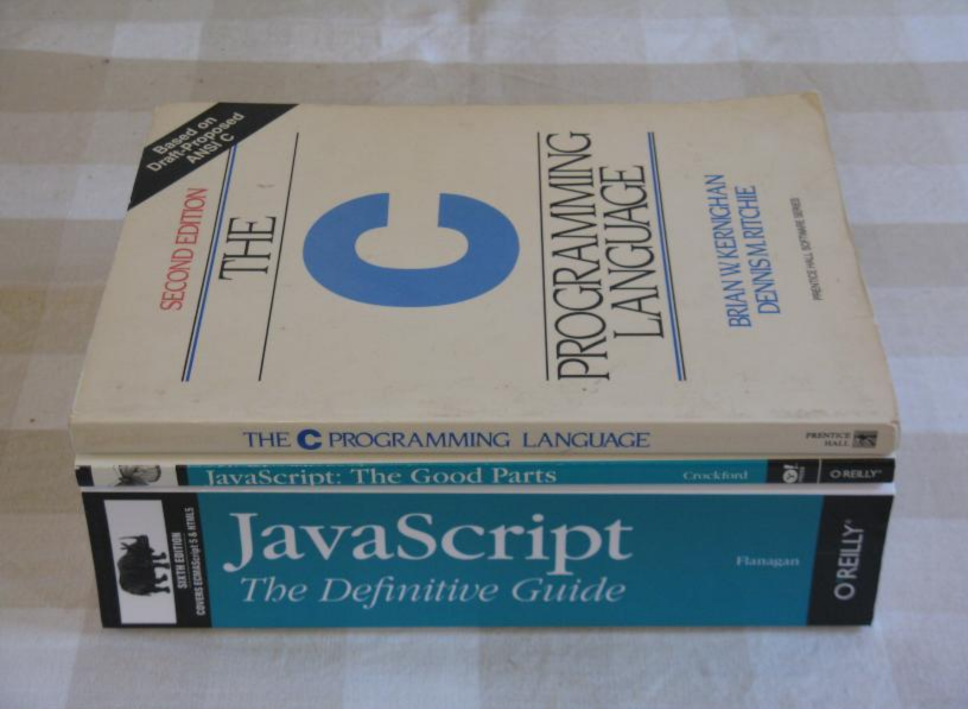
- For performance, use `std::vector<>`
- For performance, use `std::vector<native>`
- For portable performance, adapt `std::vector<native>`

Conclusions

- For performance, use `std::vector<>`
- For performance, use `std::vector<native>`
- For portable performance, adapt `std::vector<native>`
- The expressiveness of C++ can sugar this pill

Conclusions

- For performance, use `std::vector<>`
- For performance, use `std::vector<native>`
- For portable performance, adapt `std::vector<native>`
- The expressiveness of C++ can sugar this pill
  - ...or JavaScript ;-)


Based on Draft-Proposed ANSI C
SECOND EDITION
THE
C
PROGRAMMING LANGUAGE
BRIAN W. KERNIGHAN
DENNIS M. RITCHIE
PRENTICE HALL SOFTWARE SERIES
THE C PROGRAMMING LANGUAGE
PRENTICE HALL
JavaScript: The Good Parts
Crockford
O'REILLY
SIXTH EDITION
COVERS ECMAScript 5 & HTML5
JavaScript
The Definitive Guide
Flanagan
O'REILLY

Questions?

Some resources

- Herb Sutter
  - http://www.gotw.ca/publications/concurrency-ddj.htm
  - http://herbsutter.com/welcome-to-the-jungle/
- Tony Albrecht
  - http://www.slideshare.net/EmanWebDev/pitfalls-of-object-oriented-programminggcap09
- Compiler vectorization
  - https://gcc.gnu.org/projects/tree-ssa/vectorization.html
  - http://llvm.org/docs/Vectorizers.html
- OpenCL tutorial
  - http://handsonopencl.github.io/
- Agner Fog's x86 Optimization Guides
  - http://www.agner.org