STEPANOV'S LITMUS TEST IN A SCARY, HETEROGENEOUS WORLD

Alexandru Voicu
Software Engineer
Visual C++ Libraries

Our mission: `max`, `swap` and `find`

1. Generic, battle-tested, low-fat, high protein
2. Correct
3. Heterogeneity-aware

max || min

THE BOSS

We need a something that, when given two something somethings and returns The Thing! This is a key feature that will create great aligned synergies and demonstrate our continued commitment to things and stuff.

`max || min` (I)

YOU, he who knows **`<algorithm>`**

This seems like a job for **`std::max`** *or* **`std::min`***! Can haz raise☺?*

THE BOSS

Actually, what I want is a GPGPU heterogeneous distributed multi-parallel accelerated function that returns the extremum by rapport with a custom comparator, and can run on the thousands of millions of cores of the latest GeForce Radeon Xeon Titan of Braavos (of which I've ordered 10). By the way, **`std::max`** *is wrong ☹.*

`max || min` (II)

YOU, he who knows **`<algorithm>`**, worshipper at the altar of std::

I'm sure this can be done using standard C++, let me just grab my copy of Super Effective Efficient Pragmatic Bombastic Supercalifragilistic C++!

BEWARE REALITY, FOR IT IS DARK AND FULL OF HARDWARE

Once upon a time in Bell Labs..

Once upon a time in Bell Labs…

- Dennis Ritchie (Von Neumann?) opted for a great abstract machine model for C:
  - 1 processor
  - 1 flat memory pool
  - SISD execution
  - details

Decades later…

- The C and C++ abstract machine models are augmented to:
  - 1 processor (possibly with multiple homogeneous cores)
  - 1 flat memory pool
  - SISD or MIMD execution
  - a formal memory model
  - details

Reality

- Is a harsh mistress:
  - Many, heterogeneous, processors, in the same machine (CPUs, GPUs, DSPs, etc.)
  - Processors and their memory pools are distributed (and frequently incoherent)
  - MIMD and SIMD (even better if SIMT nee SPMD) execution
  - Multi-layered, potentially odd, memory model(s)
  - Lots of details

Meanwhile, in `max` || `min` land

YOU, he who knows `<algorithm>`, worshipper at the altar of `std::`, discoverer of reality

I'm sure that even though this is not in `std::` *yet, there are thousands of solutions, 3rd party libraries and extensions out there, Google walk with me!*

ALL YOU NEED IS...

The IHV

- "do everything, we are unique from head to toe and nothing that came before applies" :
  - CUDA
  - OpenCL
  - DirectCompute
- Verbose, GPU-centric, focused on exposing innards with impunity:
  - SIMD execution width under programmer control
  - Intra-SIMD thread barrier constructs (needed to maintain the above illusion, they have a FUNDAMENTAL impact on the programming model)
  - Liberal exposure of IHV-specific details
  - Composability, mainstream appeal, portability – what's that?
- Pretty good for generating vendor lock-in and job security for the enlightened

The “good-guy” librarian

- “flip a magical switch in your code and leave it to the professionals, it’s what they do”
- Most important representative, the Parallel STL proposal which is in TR nowadays
- Can appear like the bee’s knees to a particular demographic but:
  - Remember std::max?
  - std::search is $O(\ldots)$?
  - `make_pair(min_element(first, last), max_element(first, last)) == minmax_element(first, last)`?
- Black box “magic” needed to extend to other classes of processors and topologies
- Would the STL have been possible if exploratory work was impossible for anyone but the enlightened?

The “good-guy” compiler writer

- “flip a magical switch at compile-time and be amazed”
- What we usually think it means is that foo(…), the 1000 LOC monster, will be parallelized across cores and vectorised across the SIMD lanes within those cores
- What it actually means is that:

```
void foo(const int* A, const int* B, int* C, std::size_t n) {
    for (std::size_t i = 0; i != n; ++i) C[i] = A[i] + B[i];
}
```

might be parallelised and vectorised is it’s aliasing free, the stars align and Khal Drogo returns.

- The compiler is more Rincewind than it is Gandalf, don’t throw the guys writing it under a bus
- Hilariously (intractably) difficult issues once scope is expanded to account for heterogeneity

A humble programmer's perspective (no, not Dijkstra's)

- The abstract C++ machine model needs to be extended (within reason):
  1. Express locus of execution (this CPU, the GPU attached to that NUMA node etc.)
  2. Express modus of execution (SISD thread, SIMD thread etc.)
  3. Account for potentially non-uniform memory between loci
- Magical libraries should be implementable wholly within the confines of the language and the standard library i.e. it should be possible for an Ada programmer to show up and write, from scratch, the fastest parallelised and vectorised sort function that scales across NUMA nodes, abaci or what have you
- The changes and cognitive overload they bring should be minimised

Let's look at some common wisdom…

CPU

- Low to medium levels of parallelism:

$$1 \leq core_{cnt} \leq 18$$
$$1 \leq thread_{cnt} \leq 36$$

- Complicated data-structures, abstraction, algorithms are all OK
- Mainstream programming (C++, Java) – just write stuff and it works great!
- Everybody understands everything
- Everybody cares

GPU

- COLLOSAL levels of parallelism:

$$10^2 \leq core_{cnt} \leq 10^3$$
$$10^3 \leq thread\_cnt < \infty$$

- Only arrays and loops, no abstraction, extremely verbose and low-level
- Only gurus can program (OpenCL, DirectCompute) – mysterious incantations!
- Only a few gurus understand
- Only a few (misguided) gurus care

...which is wrong

CPU

- Medium levels of parallelism (if one does not ignore AVX, Neon etc.):

$$1 \leq core_{cnt} \leq 18$$
$$1 \leq SIMD_{thread_{cnt}} \leq 36$$
$$4 \leq SIMD_{width} \leq 16$$

- Complicated data-structures, abstraction, algorithms are all OK
- Mainstream programming (C++, Java) – write great code and it works great!
- Those who understand write great code

GPU

- Medium to high levels of parallelism (if one ignores marketing nonsense):

$$2 \leq core_{cnt} \leq \sim 70$$
$$8 \leq SIMD_{thread_{cnt}} < 10^3$$
$$4 \leq SIMD_{width} \leq 64$$

- Non-trivial data-structures, abstraction, algorithms are just as desirable
- Mainstream programming (C++ AMP today, ISO C++2x, Java 9 tomorrow) – write great code and it works great!
- Those who understand write great code

The desirable abstract machine model

We program processors:

1. With a variable number of cores
2. With potentially distributed memory pools
3. Capable of executing SIMD threads
4. Which follow the Von Neumann paradigm

Everything else is business as usual:

1. The interaction with memory matters for performance– locality, regularity etc.
2. "It's all about algorithms and data-structures!" *Alexander Stepanov*

*This is necessary and sufficient understanding for efficiently programming {C, G, A, *}PUs!*

Unfortunately

- We cannot get the model that we want (yet)
- So we'll approximate it on top of C++ AMP:
  - + we can experiment with CPUs and GPUs
  - + on a number of operating systems (Windows, Linux, OS X)
  - - we cannot experiment with more exotic hardware (DSPs, clusters)
  - - we have to accept some rough edges

YOU, he who knows `<algorithm>`, worshipper at the altar of `std::`, accepter of reality, forward-looking C++er

I'd better start working on that thing THE BOSS wanted!

FINALLY, SOME CODE

max (I)

```
template<typename T, typename C>
const T& max(const T& x, const T& y, C cmp) restrict(amp)
{
    return cmp(y, x) ? x : y;
}
```

max (II)

```
int main()
{
    int x = 1, y = 2;
    array_view<const int> x_av(1, &x);
    array_view<const int> y_av(1, &y);
    array_view<int> M(1);
    parallel_for_each(accelerator().default_view, M.extent,
                      [=](auto&& idx) restrict(amp) {
        M = max(x_av[0], y_av[0], [](int a, int b) { return a<b; });
    }
}
```

...

YOU

I've come up with a reasonably nice solution. Took me a while to understand all the imp…

THE BOSS

Yes yes, that is nice but we are de-emphasising and strategically re-aligning our efforts with new market realities spawned by unforeseen dynamics. The fate of our great team depends on swapping thingamebobs in parallel, fully utilising available hardware! By the way, why didn't you copy this from an FOSS project or just use the Standard Library☹.

swap (I)

```
template<typename T>
void swap(T& x, T& y) restrict(amp)
{
    T z = y;
    y = x;
    x = z;
}
```

swap (II)

```
int main()
{
    constexpr int SIMD_width = 42;
    array_view<int> X(SIMD_width); generate_n(X.data(), X.data() + SIMD_width, rand);
    array_view<int> Y(SIMD_width); generate_n(Y.data(), Y.data() + SIMD_width, rand);
    parallel_for_each(X.extent.tile<SIMD_width>(), [=](auto&& tidx) restrict(amp) {
        swap(X[tidx.local[0]], Y[tidx.local[0]]);
    }
}
```

...

YOU

One race-free, heterogeneous, parallel and vectorised **`swap`** *on aisl…*

THE BOSS

Didn't I ask for linear search! Swapping is not our core expertise anyway☹.

find

Let’s see some real code!

A happy ending

YOU

We now have a pretty reasonable **`find`** *in our codebase, I'm thinking about putting together a library and some proposals for the standard!*

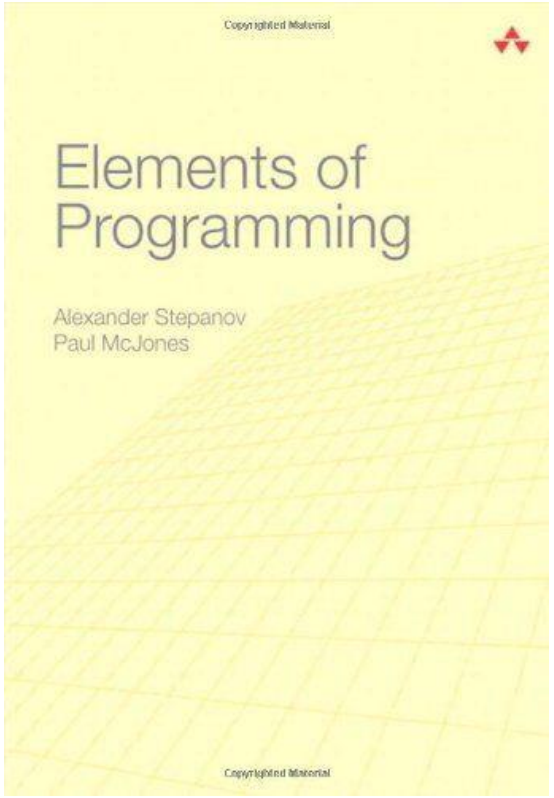
THE BOSS

Conclusions

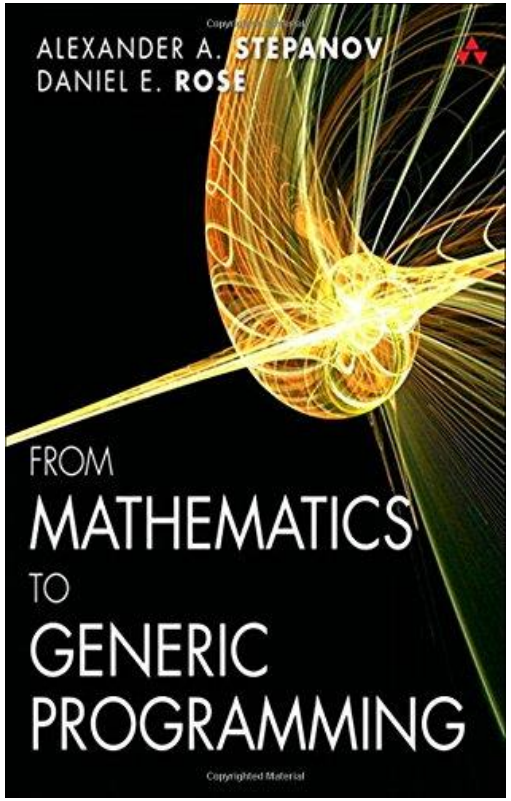
1. If we take standard C++ out of its multi-decade kennel, it has trouble finding its footing – Stepanov's test on a GPU using just the language and `std::` is a bridge too far
2. The changes necessary to "correct" (1) are far less pervasive than you'd expect from looking at GPU "languages"
3. The changes necessary to "correct" (1) involve more than a flag, tag or switch, if they are to be anything more than a crutch
4. You should start caring about heterogeneity today!

Recommended reading I

EXCELLENT BOOK ON PROGRAMMING

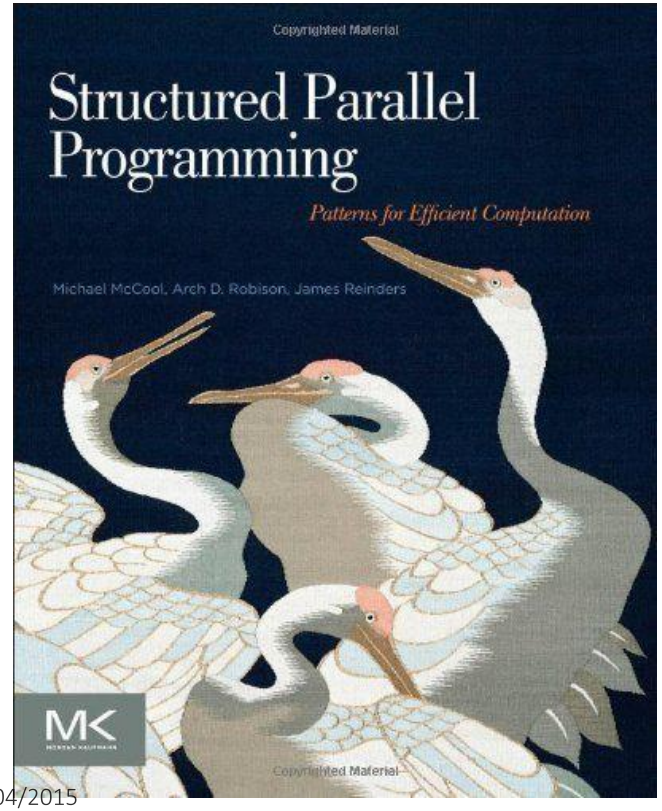


EXCELLENT BOOK ON PROGRAMMING

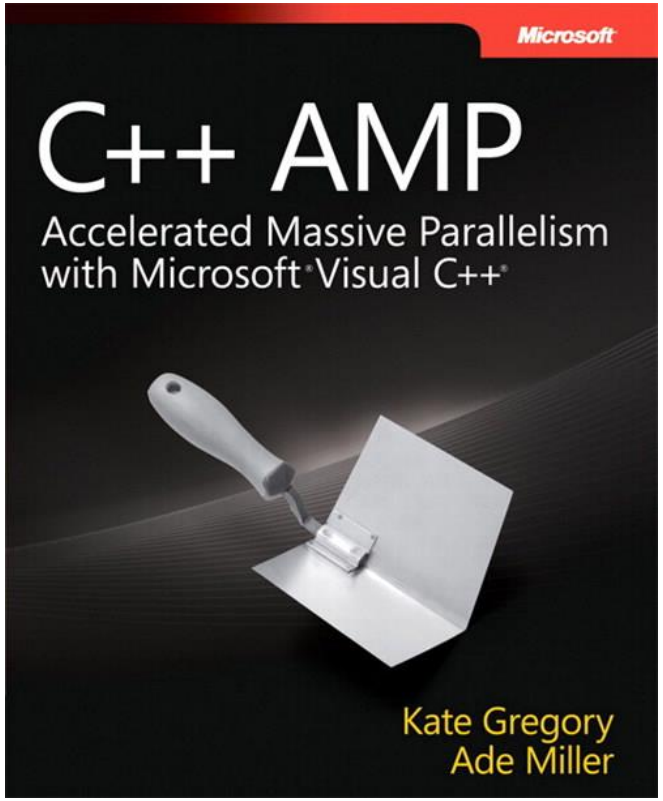


Recommended reading II

GOOD BOOK ON PARALLEL PROGRAMMING



GOOD BOOK ON PARALLEL PROGRAMMING WITH C++ AMP



Toys

You can see a further development of the ideas presented herein on (pick the descriptively named dysfunctional branch):

https://ampalgorithms.codeplex.com/

Note that it is rough – pick master (which is handled by Ade Miller) if you're not feeling adventurous but still feel like looking / playing with algorithms for your parallel, SIMD capable, CPU and/or your GPU.


Thank you

“Lasciate ogni speranza, voi ch’entrate!”