ACCELERATING CRYO-EM RECONSTRUCTION IN RELION FOR X86 & SIMD

Erik Lindahl, Stockholm University & KTH, Sweden
Charles Congdon, Intel Corporation
Outline

Our goal: Speed up scientific production codes not by 10%, but by 10x
Cryo-EM: A cool experimental technique with computational bottlenecks
Existing algorithms only reach a tiny fraction of hardware peak
Algorithmic redesign: Make the problem suitable for modern hardware
Software porting & optimization: Make new algorithms perform well on x86
Performance, portability, maintainability: Pick 3 out of 3
Lessons for design, refactoring, and maintenance of old and new code
The Nobel Prize in Chemistry 2017

Jacques Dubochet
Université de Lausanne

Joachim Frank
Columbia University

Richard Henderson
MRC Laboratory of Molecular Biology

"for developing cryo-electron microscopy for the high-resolution structure determination of biomolecules in solution"
Electron microscopy of biological samples
Structure determination of biological samples

Crystallography

Electron microscopy

Images: Wikimedia Commons (CC-BY-SA-3.0), Wikimedia Commons (PD), Johan Jarnestad/Royal Swedish Academy of Sciences, Greg Pintilie
State of the art change in cryo-EM

Classical Detector

Direct Detector Device

Motion-correction

Low damage, high motion

High damage, low motion

Radiation-weighting

Carroni, Methods 95, 78 (CC-BY-4.0)

Scheres (2016) eLife 3, e03665 (CC-BY-4.0)
RELION: A Bayesian Approach to Optimization

\[ p(A|B)p(B) = p(B|A)p(A) \]

\[ p(A|B) = \frac{p(B|A)p(A)}{p(B)} \]

This way, we can include all microscope CTFs, noise estimates and possible orientational bias in the model, as well as the assumption of smoothly varying electron density.

You need a starting model

Sjors Scheres, J Mol Biol 415, 406 (CC-BY)
Alignment & Classification

Picking & extraction → FFT

FFT

Calculate weights

Extract slice

Inverse slice back in 3D transform

Inverse FFT & iterate

Images: Kimanius et al., eLife 2016;5:e18722 (CC-BY-4.0)
Parallelization

We scan many independent
- Views (100,000s)
- Objects (10)
- Pixels (0.25 Mpix)

Images: Kimanius et al., eLife 2016;5:e18722 (CC-BY-4.0)
It is critical that you understand and re-implement the original physical model algorithm, rather than just move around or tune code. Being able to stream through data is critical!
Good Kernels Express Parallelism, not Architecture

```cpp
template <typename T>
__global__ void cuda_translate3D( T * g_image_in,
       T * g_image_out,
       int image_size,
       int xdim,
       int ydim,
       int zdim,
       int dx,
       int dy,
       int dz)
{
    int tid = threadIdx.x;
    int bid = blockIdx.x;

    int x,y,z,xp,yp,zp,xy;
    int voxel=tid + bid*BLOCK_SIZE;
    int new_voxel;

    int xxdim = xdim*ydim;
    if(voxel<image_size)
    {
        z = voxel / xxdim;
        zp = z + dz;
        xy = voxel % xxdim;
        y = xy / xdim;
        yp = y + dy;
        x = xy % xdim;
        xp = x + dx;
        if( xp>=0 && yp>=0 && xp<xdim && zp<zdim && yp<ydim && xp<xdim)
        {
            new_voxel = zp*xxdim + yp*xdim + xp;
            if(new_voxel >= 0 && new_voxel < image_size)
                g_image_out[new_voxel] = g_image_in[voxel];
        }
    }
}
```

```cpp
template <typename T>
void cpu_translate3D(T * g_image_in,
       T * g_image_out,
       int image_size,
       int xdim,
       int ydim,
       int zdim,
       int dx,
       int dy,
       int dz)
{
    int x,y,z,xp,yp,zp,xy;
    int new_voxel;

    for(int voxel=0; voxel<image_size; voxel++)
    {
        int xxdim = xdim*ydim;
        z = voxel / xxdim;
        zp = z + dz;
        xy = voxel % xxdim;
        y = xy / xdim;
        yp = y + dy;
        x = xy % xdim;
        xp = x + dx;
        if( xp>=0 && yp>=0 && xp<xdim && zp<zdim && yp<ydim && xp<xdim)
        {
            new_voxel = zp*xxdim + yp*xdim + xp;
            if(new_voxel >= 0 && new_voxel < image_size)
                g_image_out[new_voxel] = g_image_in[voxel];
        }
    }
}
```
Single precision

• Most modern CPUs provide *twice* the throughput in single compared to double
  • This also saves cache, memory bandwidth – and RELION is quite memory-hungry

• Do you need more than 7 valid digits in the output?

• Converting scientific code to single-precision is not trivial: Most codes fail if you just do search & replace, but it can be made to work:
  • Identify sensitive parts (maximization step), and leave those in double
  • Only perform a few operations (exponentials) in double
  • Sum small numbers first, or use tree summation instead of linear order
  • Use strength-reduction algorithms (check open source math libraries)

• Beware of double-single-double conversions inside critical code paths
Quality: Single precision RELION is just as accurate

Images: Kimanius et al., eLife 2016;5:e18722 (CC-BY-4.0)
“Ninja Coding”: GROMACS vs. RELION

<table>
<thead>
<tr>
<th></th>
<th>Lines of raw SIMD code</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROMACS</td>
<td>~600,000</td>
</tr>
<tr>
<td>RELION</td>
<td>0</td>
</tr>
</tbody>
</table>

Used to be raw assembly, but now intrinsics

- It is extremely likely that we could get even better speed-ups in RELION with manual tweaking, but if the compiler performs this well with only hints, why bother?
- There is a huge advantage in only having a single code path. All modifications introduced by anybody will now be x86-accelerated, even if they do not understand SIMD
## Current hardware limits - memory

We need about $8 \cdot x^3 \cdot 10^{-8}$ GB per half-reconstruction, where $x =$ pixel-size of input images.

Memory is actively used with high bandwidth, so it must stay resident on the compute device. Using the above formula, this limits us to approximately the following image sizes based on available compute device memory:

<table>
<thead>
<tr>
<th>Compute Unit</th>
<th>Particle Image Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1x NVIDIA® GeForce GTX 1080 GPU* (8GB)</td>
<td>360-pixel images</td>
</tr>
<tr>
<td>2x NVIDIA® Telsa® P100 GPU* (16GB)</td>
<td>570-pixel images</td>
</tr>
<tr>
<td>2x NVIDIA® Quadro P6000 GPU* (24GB)</td>
<td>660-pixel images</td>
</tr>
<tr>
<td>Server with 256GB</td>
<td>1100-pixel images</td>
</tr>
<tr>
<td>Server with 1TB</td>
<td>1800-pixel images</td>
</tr>
</tbody>
</table>

*Other names and brands may be claimed as the property of others

**Disclaimers:**
- Server upper limits are projected numbers, not experimentally measured
- Late-2017: Particle images over 1000 pixels are not common
Tuning for Performance

Understand application and system characteristics before diving deeper

- Use appropriate tools at each level
- Use **representative** workloads which repeatedly give the same results, and do the same work, each time executed

- **System Level**
  - Processor / Memory
  - Network / MPI
  - Disk
  - Operating System

- **Application Level**
  - Algorithmic issues
  - APIs
  - Locks
  - Heaps / Memory
  - Execution Threads

- **Micro-architecture Level**
  - Processor Stalls
  - Branch prediction
  - Code/data alignment
  - Cache optimization
  - Instructions/SIMD

*Other names and brands may be claimed as the property of others*
Data to collect about any cluster application using a wide range of data sets

• How does the runtime scale with the number of MPI ranks?
  • Both overall and the number of MPI ranks per compute node
• How does the runtime scale with the number of threads per MPI rank?
• How does the runtime scale with the amount of work done by each MPI rank?
• What parts of your code account for the most runtime change with each data set?
  • What are the bottlenecks in these parts of your code?
  • Are some of these functions unexpected (malloc, array operators, etc.)?

Understanding the algorithm is not the same as understanding what the algorithm implementation does with real data on real hardware
Some common issues seen in Cryo-EM code that harm performance

- Large amounts of serial work, I/O, or communication – limits benefits of optimizations, doesn’t effectively use compute resources
- Fine-grained parallelism – increases overhead of parallel runtime - slows execution
- Synchronization, especially in the inner loops – high overhead - serializes execution
- Unpredictable memory accesses – bypass the benefits of and reason for a cache architecture
- Too few computes per memory access – application becomes memory-bound
- Memory allocations/deallocations/memory copies, especially in inner loops (especially common in Object-Oriented code) – serializes execution
- Using double-precision when single or mixed will do – uses more memory, fetches return less, vector instructions do less per instruction
- Failure to vectorize (instruction-level parallelism) – use fraction of processor – read compiler reports!

*Other names and brands may be claimed as the property of others
Additional optimizations done for CPU on RELION

• Shared a large amount code between accelerated CPU and CUDA versions, including the mixed-precision approach discussed above
  • Non-shared code does the same thing in both implementations, only optimized for the underlying hardware
    • For example, vectorization improvements of key CPU kernels

• Parallel FFT operations are allowed in serial parts of the code

• Improved FFT performance (especially parallel FFT) with Intel® Math Kernel Library (Intel® MKL)

• Used Intel® Threading Building Blocks (Intel® TBB)’s runtime to better distribute parallel work

• Added additional parallelization

• Rearranged loops, data structures, and other logic to improve memory locality and memory access patterns

• Created lookup tables to reduce computations related to translations in Fourier space

*Other names and brands may be claimed as the property of others
Early benchmarks show order-of-magnitude acceleration

Single-node and multi-node performance of CPU and GPU accelerated code approximately identical with modern hardware

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks. Source: Measured by Intel as of November 2017. Please see detailed configuration and run information at the end of this presentation.
Conclusions

- Performance parity! Remember the real goal is performance, not hardware theoretical flops
  - Same code for Intel® Xeon® & Intel Xeon Phi™ processors - acceleration done by Intel® Parallel Studio
- RELION is improving by leaps and bounds, but there is still more that could be done
  - Every optimization just uncover new bottlenecks
  - Classic example of Amdahl's law: runtime is now dominated by parts of code that used to take single-digit percentages of runtime, which were not even multithreaded or balanced before
  - Challenging to minimize function duplication in optimized code paths for different architectures
- Accelerated CPU path is not a third new code path, but our fully portable default path.
  - Outstanding Intel® Xeon® & Intel Xeon® Phi™ processor performance with the Intel® C++ compilers, Intel® MKL, and Intel® TBB
  - Huge advantage of open standards: C++11 code will work anywhere, now and in the future

Accelerated RELION performance available for any cluster (or laptop!) in RELION-2.x
Starting closed alpha testing today, public beta before RELION-2.x release
Acknowledgements

This work would not have been possible without the support and work of the following:

• Sjors Scheres, MRC Laboratory of Molecular Biology, Cambridge, UK
• Erik Lindahl, Stockholm University & KTH, Stockholm, Sweden
• Dari Kimanius and Bjoern Forsberg, Department of Biochemistry and Biophysics, Science for Life Laboratory, Stockholm University, Stockholm, Sweden
• Kristina Kermanshahche, Intel Corporation, Oregon, USA
• Charles Congdon, Intel Corporation, Oregon, USA
• Sheng Fu, Intel Corporation, California, USA
• Yuping Zhao, Intel Corporation, Beijing, PRC

*Other names and brands may be claimed as the property of others.
Legal Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at [intel.com].

Intel, the Intel logo, Intel® Math Kernel Library (Intel® MKL), Intel® Threading Building Blocks (Intel® TBB), Intel® Xeon® & Intel Xeon Phi™ processors, Intel® Parallel Studio, Intel® C++ compilers, Intel® Xeon® processor E5-2697 v4, and Intel® Xeon® Gold 6148 processor are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit [www.intel.com/benchmarks].


NVIDIA TESLA P100*: Dual-Socket Intel® Xeon® processor E5-2697 v4 2.3 GHz, 18 Cores/Socket, 36 Cores, 72 Threads (HT and turbo on), DDR4 128GB, 2400 MHz, NVIDIA 16GB Tesla* P100 GPUs, NVIDIA CUDA* 8.0 (375.20).

NVIDIA TESLA P100 x4: Dual-Socket Intel® Xeon® processor E5-2697 v4 2.3 GHz, 18 Cores/Socket, 36 Cores, 72 Threads (HT and turbo on), DDR4 128GB, 2400 MHz, 4 NVIDIA 16GB Tesla* P100 GPUs, NVIDIA CUDA* 8.0 (375.20).

INTEL® XEON® PROCESSOR E5-2697 v4: Dual-Socket Intel® Xeon® processor E5-2697 v4 2.3GHz, 18 Cores/Socket, 36 cores, turbo and HT on, 128GB total memory, 8 slots / 16GB / 2400 MT/s / DDR4 RDIMM

INTEL® XEON® GOLD 6148 PROCESSOR: Dual-Socket Intel® Xeon® Gold 6148 processor, 2.4GHz, 40 cores, turbo and HT on, 192GB total memory, 12 slots / 16 GB / 2666 MT/s / DDR4 RDIMM

*Other names and brands may be claimed as the property of others.

Copyright © Intel Corporation
Optimization Notice

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel.

Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice Revision #20110804
**Configuration Details**

**Configuration details: RELION**

RELION: Development Build: November 7, 2017 commit on the "cpu-cuda-merge" branch of https://bitbucket.org/tcblab/relion-devel-tcblab repository (roughly equivalent to RELION 2.1 beta). This version uses a mix of single and double-precision where appropriate. The Plasmodium ribosome workload is available from [http://www2.mrc-lmb.cam.ac.uk/relion/index.php/Benchmarks_%26_computer_hardware](http://www2.mrc-lmb.cam.ac.uk/relion/index.php/Benchmarks_%26_computer_hardware)


**NVIDIA TESLA P100**: Dual-Socket Intel® Xeon® processor E5-2697 v4 2.3 GHz, 18 Cores/Socket, 36 Cores, 72 Threads (HT and turbo on), DDR4 128GB, 2400 MHz, Red Hat 7.2, Super Micro® SuperServer 1028GR-TR, Bios Version 2.0a, Super Micro® X10DRG-H Motherboard, CSE-118GHTS-R1K66BP FRU, 500GB SATA Seagate® ST9500423AS System Disk, NVIDIA 16GB Tesla® P100 GPUs, NVIDIA CUDA® 8.0 (375.20).

**NVIDIA TESLA P100 x4**: Dual-Socket Intel® Xeon® processor E5-2697 v4 2.3 GHz, 18 Cores/Socket, 36 Cores, 72 Threads (HT and turbo on), DDR4 128GB, 2400 MHz, Red Hat 7.2, Super Micro® SuperServer 1028GR-TR, Bios Version 2.0a, Super Micro® X10DRG-H Motherboard, CSE-118GHTS-R1K66BP FRU, 500GB SATA Seagate® ST9500423AS System Disk, 4 NVIDIA 16GB Tesla® P100 GPUs, NVIDIA CUDA® 8.0 (375.20).

**INTEL® XEON® PROCESSOR E5-2697 v4**: Dual-Socket Intel® Xeon® processor E5-2697 v4 2.3GHz, 18 Cores/Socket, 36 cores, turbo and HT on, SuperMicro® X10DRG-H Motherboard with American Microtrends Inc.® BIOS 2.0a, 128GB total memory, 8 slots / 16GB / 2400 MT/s / DDR4 RDIMM, 1 x 1TB SATA, Red Hat Enterprise Linux® 7.2 kernel 3.10.0-327.

**INTEL® XEON® GOLD 6148 PROCESSOR**: Dual-Socket Intel® Xeon® Gold 6148 processor, 2.4GHz, 40 cores, turbo and HT on, BIOS 86B.01.00.0412, 192GB total memory, 12 slots / 16 GB / 2666 MT/s / DDR4 RDIMM, 1 x 800GB Intel® SSD SC2BA80, Red Hat Enterprise Linux® 7.2 kernel 3.10.0-327.
Command Line Details

**Dual-socket Intel® Xeon® processor E5-2697 v4 + NVIDIA® Tesla P100 GPU** Runs:

```bash
mpirun -np 5 -perhost 5 relion_refine_mpi --o Results/gpu --i Particles/shiny_2sets.star --ref emd_2660.map:mrc --firstiter_cc --ini_high 60 --ctf --ctf_corrected_ref --tau2_fudge 4 --particle_diameter 360 --K 6 --flatten_solvent --zero_mask --oversampling 1 --healpix_order 2 --offset_range 5 --offset_step 2 --sym C1 --norm --scale --random_seed 0 --dont_combine_weights_via_disc --iter 25 --pool 100 --j 4 --gpu

mpirun -np 20 -perhost 5 relion_refine_mpi --o Results/gpu --i Particles/shiny_2sets.star --ref emd_2660.map:mrc --firstiter_cc --ini_high 60 --ctf --ctf_corrected_ref --tau2_fudge 4 --particle_diameter 360 --K 6 --flatten_solvent --zero_mask --oversampling 1 --healpix_order 2 --offset_range 5 --offset_step 2 --sym C1 --norm --scale --random_seed 0 --dont_combine_weights_via_disc --iter 25 --pool 100 --j 4 --gpu
```

**Dual-socket Intel® Xeon® processor E5-2697 v4 + NVIDIA® Tesla P100 GPU** x 4 Run:

```bash
mpirun -np 5 -perhost 5 relion_refine_mpi --o Results/gpu4 --i Particles/shiny_2sets.star --ref emd_2660.map:mrc --firstiter_cc --ini_high 60 --ctf --ctf_corrected_ref --tau2_fudge 4 --particle_diameter 360 --K 6 --flatten_solvent --zero_mask --oversampling 1 --healpix_order 2 --offset_range 5 --offset_step 2 --sym C1 --norm --scale --random_seed 0 --dont_combine_weights_via_disc --iter 25 --pool 100 --j 4 --gpu
```

*Other names and brands may be claimed as the property of others*
Command Line Details

Dual-Socket Intel® Xeon® Gold 6148 processor Runs

```sh
export OMP_SCHEDULE="dynamic"
export KMP_BLOCKTIME=0
relion_refine --o Results/skx --i Particles/shiny_2sets.star --ref emd_2660.map:mrc --firstiter_cc --ini_high 60 --ctf --ctf_corrected_ref --tau2_fudge 4 --particle_diameter 360 --K 6 --flatten_solvent --zero_mask --oversampling 1 --healpix_order 2 --offset_range 5 --offset_step 2 --sym C1 --norm --scale --random_seed 0 --dont_combine_weights_via_disc --preread_images --iter 25 --pool 160 --j 80 --cpu
```

```sh
export OMP_SCHEDULE="dynamic"
export KMP_BLOCKTIME=0
mpirun -perhost 4 -np 8 relion_refine_mpi --i Particles/shiny_2sets.star --ref emd_2660.map:mrc --firstiter_cc --ini_high 60 --dont_combine_weights_via_disc --ctf --ctf_corrected_ref --tau2_fudge 4 --particle_diameter 360 --K 6 --flatten_solvent --zero_mask --oversampling 1 --healpix_order 2 --offset_range 5 --offset_step 2 --sym C1 --norm --scale --random_seed 0 --o Results/cpu --pool 40 --j 20 --iter 25 --cpu
```

```sh
export OMP_SCHEDULE="dynamic"
export KMP_BLOCKTIME=0
mpirun -perhost 4 -np 16 relion_refine_mpi --i Particles/shiny_2sets.star --ref emd_2660.map:mrc --firstiter_cc --ini_high 60 --dont_combine_weights_via_disc --ctf --ctf_corrected_ref --tau2_fudge 4 --particle_diameter 360 --K 6 --flatten_solvent --zero_mask --oversampling 1 --healpix_order 2 --offset_range 5 --offset_step 2 --sym C1 --norm --scale --random_seed 0 --o Results/cpu --pool 40 --j 20 --iter 25 --cpu
```

```sh
export OMP_SCHEDULE="dynamic"
export KMP_BLOCKTIME=0
mpirun -perhost 4 -np 32 relion_refine_mpi --i Particles/shiny_2sets.star --ref emd_2660.map:mrc --firstiter_cc --ini_high 60 --dont_combine_weights_via_disc --ctf --ctf_corrected_ref --tau2_fudge 4 --particle_diameter 360 --K 6 --flatten_solvent --zero_mask --oversampling 1 --healpix_order 2 --offset_range 5 --offset_step 2 --sym C1 --norm --scale --random_seed 0 --o Results/cpu --pool 40 --j 20 --iter 25 --cpu
```
Command Line Details

Dual-Socket Intel® Xeon® processor E5-2697 v4 Optimized Runs
export OMP_SCHEDULE="dynamic"
export KMP_BLOCKTIME=0
relion_refine --o Results/bdw --i Particles/shiny_2sets.star --ref emd_2660.map:mrc --firstiter_cc --ini_high 60 --ctf --ctf_corrected_ref --tau2_fudge 4 --particle_diameter 360 --K 6 --flatten_solvent --zero_mask --oversampling 1 --healpix_order 2 --offset_range 5 --offset_step 2 --sym C1 --norm --scale --random_seed 0 --dont_combine_weights_via_disc --preread_images --iter 25 --pool 144 --j 72 --cpu

export OMP_SCHEDULE="dynamic"
export KMP_BLOCKTIME=0
mpirun -perhost 4 --np 16 relion_refine_mpi --i Particles/shiny_2sets.star --ref emd_2660.map:mrc --firstiter_cc --ini_high 60 --dont Combine_weights_via_disc --ctf --ctf_corrected_ref --tau2_fudge 4 --particle_diameter 360 --K 6 --flatten_solvent --zero_mask --oversampling 1 --healpix_order 2 --offset_range 5 --offset_step 2 --sym C1 --norm --scale --random_seed 0 --o Results/cpu --pool 36 --j 18 --iter 25 --cpu

Dual-Socket Intel® Xeon® processor E5-2697 v4 Baseline Runs
relion_refine --o Results/gold --i Particles/shiny_2sets.star --ref emd_2660.map:mrc --firstiter_cc --ini_high 60 --ctf --ctf_corrected_ref --tau2_fudge 4 --particle_diameter 360 --K 6 --flatten_solvent --zero_mask --oversampling 1 --healpix_order 2 --offset_range 5 --offset_step 2 --sym C1 --norm --scale --random_seed 0 --dont Combine_weights_via_disc --preread_images --iter 25 --pool 10 --j 72

mpirun -perhost 4 --np 16 relion_refine_mpi --o Results/mpi --i Particles/shiny_2sets.star --ref emd_2660.map:mrc --firstiter_cc --ini_high 60 --ctf --ctf_corrected_ref --tau2_fudge 4 --particle_diameter 360 --K 6 --flatten_solvent --zero_mask --oversampling 1 --healpix_order 2 --offset_range 5 --offset_step 2 --sym C1 --norm --scale --random_seed 0 --dont Combine_weights_via_disc --iter 25 --pool 10 --j 18