Accelerating epistasis detection on Intel CPUs and discrete GPUs with Intel® Advisor

Aleksandar Ilic, Diogo Marques, Rafael Campos and Zakhar Matveev
Hardware Heterogeneity

Optimization

Application Diversity

Cache-aware Roofline
Boosting Epistasis Detection with oneAPI

GPU Optimization

CPU Optimization

Intel® Advisor

oneAPI

HiperBio

SPAR
CITY

http://sparcity.eu
Outline
Cache-aware Roofline Model

Roofline in a nutshell

Communication overlapped with computation
Max performance capped by peak compute throughput or available bandwidth (processor’s view)
What is bandwidth?

Cache-aware Roofline Model (CARM)\(^1\): Bandwidth as seen by the core
- Obtained via micro-benchmarking

Original Roofline Model (ORM)\(^2\): Bandwidth between memory levels
- Can be obtained from data-sheets

---


Implications ...

Cache-aware Roofline Model\textsuperscript{1}
- One model, one arithmetic intensity
- One application “point”

Original Roofline Model\textsuperscript{2}
- Several models, several intensities
- Several application “points”

\textsuperscript{2} S. Williams, A. Waterman, D. Patterson, ”Roofline: An Insightful Visual Performance Model for Multicore Architectures”, Commun. ACM (2009)
Implications ... bring cool features

Cache-aware Roofline Model
- Shows absolute architecture maximums*
(You can’t break them! Can your application exploit them?)

How to “plot” my code?
- CARM arithmetic intensity is exactly what you expect it to be!

Arithmetic Intensity (flops/byte)
log → Performance (flops/s)
log →

code
... float a = A[i];
float b = B[i];
float c = a*b;
C[i] = c;
...

assembly
... 
ld r1, mem[add1]
ld r2, mem[add2]
mul r3, r1, r2
st mem[add3], r3
...

counters
...
MEM_RETIRED.LOADS
MEM_RETIRED.STORES
INST_RETIRED.FLOPS
CPU_CLK_CYCLES.ALL
...

* We relaxed this requirement in our FGCS paper (2020)

Implications ... bring cool features

Cache-aware Roofline Model
- Shows absolute architecture maximums
  (You can’t break them! Can your application exploit them?)

How to “plot” my code?
- CARM arithmetic intensity is exactly what you expect it to be!

Intel Advisor Roofline feature
- CARM is there since 2017

Implications ... bring cool features

Cache-aware Roofline Model
- Shows absolute architecture maximums (You can’t break them! Can your application exploit them?)

How to “plot” my code?
- CARM arithmetic intensity is exactly what you expect it to be!

How to use CARM?
① Detect the boundness region
  - What are my expected maximums?
  - Provides first optimization hints

② Draw an imaginary vertical line
  - What are my main bottlenecks? (observe intersected lines)
  - Focus your optimization (aim at surpassing the line above)

③ Optimize your code: Break above roofs!
  - You should move up (as your performance improves)
  - Unless you restructure the code, or your compiler decides so…

Matrix Multiplication

All codes AVX vectorized!

[1] Basic implementation (row major)

\[
A \times B = C
\]

[2] Transposed B (improved mem. access)

\[
A \times B^T = C
\]

[3,4,5] Cache blocking: L3, L2, L1

\[
A \times B = C
\]

[6] Intel MKL

\[
\text{Intel MKL}
\]
Cache-aware Roofline Model: Extensions

CARM-based DVFS analysis

GPU CARM: Performance, Power, DVFS

NUMA CARM: Multi-socket, KNL

Epistasis Detection

Epistasis in a nutshell

Some SNP interactions may cause life-threatening diseases (e.g., Alzheimer, breast cancer). Discovering which and how many is important, but challenging task!
# Binarizing your genotype

<table>
<thead>
<tr>
<th>SNP X</th>
<th>P0</th>
<th>P1</th>
<th>Genotype</th>
<th>A1</th>
<th>A2</th>
<th>phenotype</th>
</tr>
</thead>
<tbody>
<tr>
<td>X0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Homozygous Major**  
**Heterozygous**  
**Homozygous Minor**

Think: Patient 1 (P1) with genotype 2 has disease (case)
Dataset structure

<table>
<thead>
<tr>
<th>SNP X</th>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>...</th>
<th>PN</th>
</tr>
</thead>
<tbody>
<tr>
<td>X0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td>X1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>X2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>

phenotype

0 1 1 1 0 0 ... 1

N Samples (Patients)
M SNPs

SNP structure

Our dataset: 10 048 SNPs x 104 448 samples
2-way Epistasis Detection: Pair-wise interaction

Pair-wise interaction: SNPs (X,Y)

Frequency table

<table>
<thead>
<tr>
<th>genotype combination</th>
<th>ph.type: 0</th>
<th>ph.type: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X1 Y0</td>
<td>X2 Y2</td>
</tr>
</tbody>
</table>

Each frequency table evaluated with Bayesian K2 score

Epistasis: Minimum K2 score among all combinations!

Dataset structure

Our dataset: 10 048 SNPs x 104 448 samples

Our dataset: 50 476 128 combinations

Search space: All SNP combinations

M(M-1)/2 combinations

Epistasis Detection: GPU Optimization
DPC++ setup

**Host code**

```c++
uint* dev_data = (uint*) cl::sycl::malloc_shared(M * N * 3 * sizeof(uint), device, context);
uint* dev_phen = (uint*) cl::sycl::malloc_shared(M * sizeof(uint), device, context);
float* dev_scores = (float*) cl::sycl::malloc_shared(M * M * sizeof(float), device, context);

for(x = 0; x < M * N * 3; x++)
    dev_data[x] = host_data[x];
for(x = 0; x < N; x++)
    dev_phen[x] = host_phen[x];

cl::sycl::range<2> global_range(M, M);
cl::sycl::range<2> local_range(WORK_GROUP_SIZE, 1);
queue.submit([&](cl::sycl::handler& h)
{
    h.parallel_for<class kernel_epi>(cl::sycl::nd_range<2>(global_range, local_range),
    [=](cl::sycl::nd_item<2> id)
    {
        // device code
    });
});
```

The kernel is launched for $M \times M$ work-items

Work-items without a valid combination will not do work
DPC++ setup

Device code

```cpp
c1::sycl::range<2> global_range(M, M);
c1::sycl::range<2> local_range(WORK_GROUP_SIZE, 1);
queue.submit([&](cl::sycl::handler& h) {
    h.parallel_for<class kernel_epi>(c1::sycl::nd_range<2>(global_range, local_range),
    [](c1::sycl::nd_item<2> id) {
        i = id.get_global_id(0);
        j = id.get_global_id(1);
        if (j > 1)
            // process combination
    });

```

The kernel is launched for \(M \times M\) work-items

Work-items without a valid combination will not do work
Pair-wise interaction: SNPs (i,j)

The frequency table is filled for SNPs i and j by going through all samples, disposed in columns.
GPU Roofline in Advisor

Summary View:

GPU Optimization

Three Genotypes + Phenotype
Advisor in action…

Performance needs improvement!
The application is memory bound – let’s restructure
Restructuring our algorithm...

Pair-wise interaction: SNPs \((i,j)\)

Reducing memory transfers!

“New” Dataset structure
(removed: phenotype and genotype 2)
**DPC++ implementation**

**Pair-wise interaction:** SNPs \((i,j)\)

The frequency table is filled for SNPs \(i\) and \(j\) is filled separately for each phenotype.

### Device code

```cpp
// phenotype zero
SNPi = &data_zeros[i * N_zeros * 2];
SNPj = &data_zeros[j * N_zeros * 2];
for (p = 0; p < 2 * N_zeros; p += 2)
{
    di2 = ~(SNPi[p] | SNPi[p + 1]);
    dj2 = ~(SNPj[p] | SNPj[p + 1]);

    ft[8] += cl::sycl::popcount(SNPi[p] & SNPj[p]);
    ft[1] += cl::sycl::popcount(SNPi[p] & SNPj[p + 1]);
    ft[2] += cl::sycl::popcount(SNPi[p] & dj2);
    ft[3] += cl::sycl::popcount(SNPi[p + 1] & SNPj[p]);
    ft[4] += cl::sycl::popcount(SNPi[p + 1] & SNPj[p + 1]);
    ft[5] += cl::sycl::popcount(SNPi[p + 1] & dj2);
    ft[6] += cl::sycl::popcount(di2 & SNPj[p]);
    ft[7] += cl::sycl::popcount(di2 & SNPj[p + 1]);
    ft[8] += cl::sycl::popcount(di2 & dj2);
}
// phenotype one...
```

---

**GPU Optimization**

- Three Genotypes + Phenotype
- Two Genotypes, No Phenotype
Advisor in action…

Changing the algorithm moved us to the left! Performance is worse!
Advisor in action…

**GPU Optimization**

- **Three Genotypes + Phenotype**
- **Two Genotypes, No Phenotype**

---

**Graphs and Data:***

- **Speedup:** 1.8x
- **ALUops:** 2.0x decrease
- **Bytes:** 1.5x decrease
- **Arith. Intensity:** 1.3x decrease

**Not so good thing!**

Your speedup better be pretty high to surpass performance of previous version (think: IPC).

**Not so good thing!**

But we do have better algorithm (speedup and performance improvement).
Let’s continue optimizing…

Three Genotypes + Phenotype

Two Genotypes, No Phenotype

Transposed Dataset

Improving memory accesses by dataset transposition

Coalescing of memory accesses by the work-items
Advisor in action…

Memory accesses are now more efficient!
We have now “block loads” versus a lot of gathers!

1.7x performance improvement
1.7x speedup

GPU Optimization

- Three Genotypes + Phenotype
- Two Genotypes, No Phenotype
- Transposed Dataset
Advisor in action…

GPU Optimization

- Three Genotypes + Phenotype
- Two Genotypes, No Phenotype
- Transposed Dataset

How can we improve this result?
Data set tiling

Device code

```c
// phenotype zero
SNPi = &data_zeros[(i_block * N_zeros + (i - i_block)) * 2];
SNPj = &data_zeros[(j_block * N_zeros + (j - j_block)) * 2];
for(p = 0; p < 2 * block_sn * N_zeros; p += 2 * B_SNP)
{
    di2 = ~SNPi[p] | SNPj[p + 1];
    dj2 = ~SNPj[p] | SNPi[p + 1];

    ft[0] += cl::sycl::popcount(SNPi[p] & SNPj[p]);
    ft[1] += cl::sycl::popcount(SNPi[p] & SNPj[p + 1]);
    ft[2] += cl::sycl::popcount(SNPi[p] & dj2);
    ft[3] += cl::sycl::popcount(SNPi[p + 1] & SNPj[p]);
    ft[4] += cl::sycl::popcount(SNPi[p + 1] & SNPj[p + 1]);
    ft[5] += cl::sycl::popcount(SNPi[p + 1] & dj2);
    ft[6] += cl::sycl::popcount(di2 & SNPj[p]);
    ft[7] += cl::sycl::popcount(di2 & SNPj[p + 1]);
    ft[8] += cl::sycl::popcount(di2 & dj2);
}
// phenotype one
...
```

Tiling our dataset to squeeze the maximums!
Using a tile size of $B_{SNP}$ we can maintain constant access stride

The data set is iterated by rows of size $2 \times B_{SNP}$.
The 3rd genotype is computed.
The frequency table is filled using 2 genotypes.

GPU Optimization

- Three Genotypes + Phenotype
- Two Genotypes, No Phenotype
- Transposed Dataset
- Dataset Tiling
Advisor in action…

Further performance increase!
The performance is now close to the compute roof!
Conclusions

Special thanks to Sujata Tibrewala, Zakhar A. Matveev and Intel Advisor team, and Igor Vorobtsov

Intel DevMesh Project - Boosting epistasis detection on Intel CPU+GPU systems | Intel DevMesh

Thank you!

Supported by Intel Corporation, Portuguese FCT national funds and EuroHPC Joint Undertaking under projects UIDB/50021/2020, PTDC/CCI-COM/31901/2017 (HiPERBio), and grant agreement no. 956213 (Sparcity)