Andrei Poenaru
University of Bristol

Performance Characterisation of HPC Mini-apps on Arm SVE

https://uob-hpc.github.io
Overview of HPC on Arm at Bristol

• Isambard is the world’s first production 64-bit Arm supercomputer
  • ~160 nodes of 32-core TX2 in Cray XC50
  • Phase 1 installed late 2017, full system late 2018

• One of the 3 universities part of the Catalyst UK project
  • 64 nodes of 32-core TX2 in HPE Apollo 70
  • Installed early 2019

• Published two performance studies, single-node [1] and at-scale [2]
Towards SVE

• Upcoming generations of Arm HPC processors will use SVE
  • Isambard 2 may include A64FX nodes
  • Beginning work *now* enables a rolling start when hardware is available

• A multi-dimensional problem:
  • Functional correctness – a lot of work already done
  • Efficacy of using SVE – the level we can best address today
  • Real performance projections – hardest to tackle, but most interesting
Tools

• Static analysis:
  • Compiler reports can help identify vectorisation issues
  • Raw assembly code sometimes shows compiler’s decisions more clearly

• Dynamic analysis:
  • ArmIE – fast enough to run most mini-apps under a reduced test case
    • But still not fast enough for others, e.g. SNAP
  • Custom instrumentation is useful to collect the relevant data
  • Post-processing needed to aggregate and filter ArmIE output
    • Can be expensive

• Simulation:
  • gem5 (sve/beta1) – simulation speed is an issue
Mini-apps

• Ideal for experiments where real hardware isn’t available
  • Configurable problem sizes with fine-grained control
  • Simple, well-understood kernels
    • Easy to quantify vectorisation efficacy
  • Many have built-in validation procedures

• STREAM, BUDE, TeaLeaf, CloverLeaf, Neutral, MiniFMM, MegaSweep
  • Cover a wide range of scientific application classes
  • The same used in TX2 papers
  • MegaSweep used because SNAP is too slow
Restrictions

• The mini-apps use a combination of MPI and OpenMP
  • Disable MPI (or run with a single rank) and run a single thread

• Run times kept very low, so that we could potentially simulate the same inputs
  • 1–5 seconds on a real TX2
  • Number of iterations can further be reduced if needed

• Disable all profiling and validation
  • Results validated separately

https://uob-hpc.github.io
Analysis (1)

1. How well is SVE targeted in compilers?
   • We have access to three SVE compilers: Arm, Cray (alpha), GNU
   • Plan to add Fujitsu soon
   • Loosely similar performance, but there are differences
   • Compare between compilers, but also between platforms (i.e. vs NEON and AVX)
2. How much SVE is executed at run-time?

- Instrument the code to provide:
  - A breakdown of SVE instructions executed, related to chosen SVE width – there is a danger that SVE code is generated but branched over
  - Real occupancy of SVE vectors – SVE code is always predicated, so it is possible that “scalar” SVE instructions are being executed
3. What (real) performance can we expect?
   • Can only answer this using a simulator
   • We are looking for correlations between data obtained from instrumentation and simulated performance
     • This is still work in progress
## Results: Vectorisation

<table>
<thead>
<tr>
<th>Application</th>
<th>% time</th>
<th># Loops</th>
<th>Loops Vectorised SVE</th>
<th>Loops Vectorised NEON</th>
<th>Loops Vectorised AVX*</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUDE</td>
<td>98.6</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>TeaLeaf (cg)</td>
<td>87.2</td>
<td>8</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>TeaLeaf (ppcg)</td>
<td>91.2</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>CloverLeaf</td>
<td>62.5</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>MegaSweep</td>
<td>70.3</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Neutral</td>
<td>85.8</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MiniFMM</td>
<td>98</td>
<td>8</td>
<td>7</td>
<td>—</td>
<td>5</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>42</td>
<td>32</td>
<td>29</td>
<td>28</td>
<td>28</td>
</tr>
</tbody>
</table>

Compiler versions used: Arm 19.2, Cray 9.0a, GCC 8.2

* No difference between AVX2 and AVX-512
Results: SVE Instruction Usage (1)

Op Group:
- A64
- arithmetic
- control
- mem-read
- mem-write
- move
- other

SVE

Dynamic execution count (instructions)

https://uob-hpc.github.io
Results: SVE Instruction Usage (2)

SVE
Results: SVE Instruction Usage (3)
Results: SVE Vector Utilisation (1)

• Most mini-apps get ideal vector utilisation (all lanes of SVE vectors are active for all operations):
  • STREAM
  • BUDE
  • TeaLeaf
  • CloverLeaf
  • MegaSweep
    • Only Arm and Cray; no vectorisation with GCC

• But there are exceptions: MiniFMM
Results: SVE Vector Utilisation (2)

fmm.omp-sve

svewidth = 128

svewidth = 256

svewidth = 512

svewidth = 1024

Arm 19.2

GCC 8.2

https://uob-hpc.github.io
Results: SVE Memory Operations (1)

BUDE

MiniFMM

Arm 19.2, 512-bit SVE

https://uob-hpc.github.io
Results: SVE Memory Operations (2)

MegaSweep

CloverLeaf

Arm 19.2, 512-bit SVE

https://uob-hpc.github.io
Lessons Learned

• We were able to run binaries produced by Arm, Cray, and GCC, both from C and Fortran

• Obtaining a full set of results requires a sequence of several operations
  • We’ve found using wrapper scripts for an additional level of abstraction helps both with collection and data organisation

• For single-core runs, aim for <10 s on a real TX2 core

• If writing custom instrumentation, consider processing as you go
  • Overhead of a clean call can justify doing extra work in the instrumentation client to avoid the need to post-process
  • Thread safety in DynamoRIO is not trivial
Challenges: Analysis Tools

• ArmIE performance is good for plain emulation, but adding instrumentation can quickly slow it down significantly

• Memory tracing produces **huge** output files
  • Can easily reach several GBs for a second of real TX2 run-time
    • MegaSweep and SNAP are very challenging here
  • Several output files produced by default, which need to be merged
  • Post-processing the output is an expensive task in itself

• We’ve encountered some hard-to-reproduce segmentation faults with custom instrumentation
  • DynamoRIO documentation leaves to be desired
  • Working with Arm to resolve the issue

https://uob-hpc.github.io
Challenges: Compilers

• Yes, we can target SVE in compilers
  • But how optimal is this generated code for a real processor?

• We know (or can sensibly guess) micro-architectural details for upcoming SVE processors
  • But there’s no way to pass them to the compilers

• For micro-architecture experiments, we would want a way to describe a hypothetical processor to the compiler
  • The same configuration can then be simulated
Further Work

• Further refining of emulation-based experiments
  • Working on applying regions of interest to all clients, so that we don’t record data in initialisation and clean-up
  • Trying to identify and fix the cause of intermittent segfaults

• We are building SimEng (“Simulation Engine”), a flexible and accurate simulation toolkit designed specifically with HPC processors in mind
  • SVE micro-architecture design-space exploration is our first goal
Questions
[1] Comparative Benchmarking of the First Generation of HPC-Optimised Arm Processors on Isambard
S. McIntosh-Smith, J. Price, T. Deakin and A. Poenaru, CUG 2018, Stockholm

[2] Scaling Results From the First Generation of Arm-based Supercomputers
S. McIntosh-Smith, J. Price, A. Poenaru and T. Deakin, CUG 2019, Montreal