ATOS introduction
ST/Linaro Collaboration Context

Presenter: Christian Bertin

Development team: Rémi Duraffort, Christophe Guillon, François de Ferrière, Hervé Knochel, Antoine Moynault

Consumer Product Division, Compilation team

2016/3/8
Project Rationale

• Optimizing large open source applications by hand is a dead-end
  ⇒ Sources and makefiles must be optimized as is

Let the machine do the hard work

(HiPEAC Roadmap)

Develop a tool, which automatically tunes the compiler for a specific set of use cases, seamlessly applying advanced compilation optimization
Auto Tuning Optimization System

ATOS stands for:

“Auto Tuning Optimization System”

• Functional requirements:
  • Automatically find best C/C++ compiler configuration for a given objective.
  • Explore on top of any build system with a given set of executable use cases
  • Preserve original sources, makefiles and build scripts (they are not modified)

• Application:
  • Find best performance / code size tradeoffs search for:
    • a given set of executables/libraries[/kernel modules] and
    • a given set of benchmarking use cases
Optimization Challenges

- WebCore
- V8 JIT
- Cairo
- Jpeg
- Pixman
- Directfb
- Gtk

- Identify Hot Code
- Advanced Profiling Tools
- Find best compiler tuning
- Enlarge Optimization scope
- Inter Procedural and Cross Library
- Iterative Optimization
ATOS High Level Usage

Developer or batch environment

Unmodified compile.sh: actually any build system command:
- make all
- rpmbuild
- build.sh

Release environment

Optimized executable

eexec.sh: actually a script running the application
- make run
- direct or remote execution
- run.sh
- emulated or on board

Unmodified compile.sh:
- atos-explore
  - unmodified compile.sh
  - exec.sh
  - executable name
  - Local or global Configuration Database

Extract of database
- atos-play
  - unmodified compile.sh
  - Optimized executable
Example of Preferred Configuration

- Best trade-off
- Best size
- Get and replay chosen configuration
ATOS Features

• Whole executable optimization sequences
• File by file exploration of optimization sequences
• Function by function exploration with provided GCC plugin
• Support for 'perf' and 'oprofile' tools
• Seamless profiling feedback and link time optimizations support
• Support in GCC 4.5 to 6.0 / LLVM 3.4 and 3.6 / ARM RVCT
• Simple command line interface
• Native or cross build and run
ATOS Features (continued)

Latest features/improvements (ATOS v4.0)

• Kernel build support (*.ko)

• Various exploration schemes: random, staged or genetic

• Parallel build and execution (native or remote) capabilities

• Parameters fine-tuning and flags pruning exploration
Example of Exploration Usage

**Initialization of session**

$ atos-init -b ./build.sh -r ./run.sh -p ./run-oprofile.sh

**First basic exploration O(12)**

$ atos-explore

**Whole program flags exploration O(M.G.3) [def: O(3000)]**

$ atos-explore-genetic [-M100] [--generations 10]

**File by file exploration O(H.N.36) [def: O(H.3600)]**

$ atos-explore-acf -file-by-file [-N100]

**Funct by funct exploration O(H.N.36) [def: O(H.3600)]**

$ atos-explore-acf [-N100]

**Show exploration graph**

$ atos-graph

Where: O(...): complexity in number of build+run, H: number of determined Hot file (resp. function)
Example of Replay Usage

**Output best perf tradeoff**

$ atos-play -T

```
speedup | sizeded | target | variant_id
+30.56% | -13.44% | sha-shatest-shacmp | bbc60eb968fb803987f4df19304e6a1e
```

**Output best size tradeoff**

$ atos-play -T -f size

```
speedup | sizeded | target | variant_id
+13.57% | +3.58% | sha-shatest-shacmp | 7b9351c456fa303f8e3ef446e2028aff
```

**Compile for best perf tradeoff**

$ atos-play

Building Variant [bbc60eb968fb803987f4df19304e6a1e]...

**Compile a selected variant**

$ atos-play -l 7b9351c456fa303f8e3ef446e2028aff

Building Variant [7b9351c456fa303f8e3ef446e2028aff]...
Video of exploration using parallel build/run features
ATOS Outcomes
ATOS Optimization Results

• JPEG ST40 / HDK7108 results
  • 26.39% speedup / 13.37% size increase

• ZLIB ST40 / HDK7108 results
  • 12.54% speedup / 1.41% size increase

• Stagecraft ST40 / HDK7108 results
  • 5-28% speedup (30 benches) / 14% size reduction

• HEVC ARMv7 / Orly results
  • 9.22% speedup / 21.21% size reduction

• SPEC2000 ARMv7 / U9540 results
  • 18.7% speedup SPECINT / 10.2% speedup SPECFP / size increase n/a
Best perf v2.0: 26.39% speedup 13.37% size increase
ZLIB ST40 / HDK7108

Best perf v2.0:
12.54% speedup
1.41% size increase
StageCraft ST40 / HDK7108

• **Results:**
  - **Performance:** 5-15 % speedup range in FPS, one at 28%
  - **Size:** 14% footprint reduction (out of 35Mb initial footprint)

• Reference: build from CPD –O2/-O3 mix, compiler gcc 4.5.2 (no lto, no plugins)
• Benchmark: Sagem provided benchmarks
• Profile: 58 use cases, 30 of which not fully HW accelerated
• Optimization scope Stagecraft binary and 21 accompanying shared objects
• Outcome: Customer interested but actually the packages are provided by ST/CPD
ATOS* HEVC** results
Summary and robustness to source updates

Speed-up and code size compared to reference configuration: -O3 build
Performance measurement unit: CPU time (process user time)
Benchmark: decoding of the 300 first frames of a typical stream***

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Performance first</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>speed-up</td>
</tr>
<tr>
<td>1. Initial exploration (HEVC version 1) (4618 runs)</td>
<td>6.18%</td>
</tr>
<tr>
<td>2. Replay best after update (HEVC version 2) (0 more run)</td>
<td>7.55%</td>
</tr>
<tr>
<td>3. More advanced fine-tuning on top of 2. (1267 additional runs)</td>
<td>9.22%</td>
</tr>
</tbody>
</table>

Source update without additional exploration

Some more explorations

* ATOS version: atos v2.0
** HEVC version: HEVC full SW ARM Cortex, Optimized, 2 distinct versions
*** stream: RA_LC_TotalRecallHotelTransylvania_1280x720_25_QP26.bin
### SPECINT2000* ARMv7 / U9540

<table>
<thead>
<tr>
<th></th>
<th>Time OS</th>
<th>Speedup -O3/-O3</th>
<th>Speedup -Os/ATOS</th>
<th>Speedup -O3/ATOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>164.gzip</td>
<td>40.9</td>
<td>7.4</td>
<td>24.8</td>
<td>16.1</td>
</tr>
<tr>
<td>175.vpr</td>
<td>37.6</td>
<td>11.7</td>
<td>16.8</td>
<td>4.6</td>
</tr>
<tr>
<td>176.gcc</td>
<td>4.0</td>
<td>6.7</td>
<td>25.4</td>
<td>17.6</td>
</tr>
<tr>
<td>181.mcf</td>
<td>45.6</td>
<td>2.2</td>
<td>5.0</td>
<td>2.8</td>
</tr>
<tr>
<td>186.crafty</td>
<td>31.7</td>
<td>8.5</td>
<td>21.1</td>
<td>11.5</td>
</tr>
<tr>
<td>197.parser</td>
<td>12.1</td>
<td>5.3</td>
<td>29.3</td>
<td>22.8</td>
</tr>
<tr>
<td>252.eon</td>
<td>12.4</td>
<td>54.7</td>
<td>80.1</td>
<td>16.4</td>
</tr>
<tr>
<td>253.perlbmk</td>
<td>68.6</td>
<td>4.2</td>
<td>22.2</td>
<td>17.3</td>
</tr>
<tr>
<td>254.gap</td>
<td>7.0</td>
<td>0.7</td>
<td>8.6</td>
<td>7.8</td>
</tr>
<tr>
<td>255.vortex</td>
<td>15.4</td>
<td>2.5</td>
<td>85.6</td>
<td>81.2</td>
</tr>
<tr>
<td>256.bzip2</td>
<td>48.2</td>
<td>3.1</td>
<td>17.7</td>
<td>14.1</td>
</tr>
<tr>
<td>300.twolf</td>
<td>17.8</td>
<td>0.3</td>
<td>28.3</td>
<td>27.8</td>
</tr>
</tbody>
</table>

| SPECint  | +8.2%  | +28.4% | +18.7% |

* SPECINT2000 results and improvements on train dataset
(Speedup -O3/ATOS: +13.97% when re-executed on REF dataset)
## SPECFP2000* ARMv7 / U9540

<table>
<thead>
<tr>
<th></th>
<th>Time -OS</th>
<th>Speedup -Os/-O3</th>
<th>Speedup -Os/ATOS</th>
<th>Speedup -O3/ATOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>177.mesa</td>
<td>47.1</td>
<td>27.9</td>
<td>54.4</td>
<td>20.7</td>
</tr>
<tr>
<td>179.art</td>
<td>22.3</td>
<td>0.3</td>
<td>9.2</td>
<td>8.9</td>
</tr>
<tr>
<td>183.equake</td>
<td>40.4</td>
<td>3.3</td>
<td>6.4</td>
<td>2.9</td>
</tr>
<tr>
<td>188.ammp</td>
<td>165.0</td>
<td>66.9</td>
<td>82.1</td>
<td>9.1</td>
</tr>
<tr>
<td>SPECfp</td>
<td></td>
<td></td>
<td>+22.0%</td>
<td>+10.2%</td>
</tr>
</tbody>
</table>

* SPECINT2000 results and improvements on TRAI dataset
(Speedup -O3/ATOS: +6.53% when re-executed on REF dataset)
Compilation Technologies for ATOS

Fine grain optimizations from whole to per file to per function

Use application training run to drive optimizations

Perform optimizations across object files

Explore the space of possible compiler optimizations mix

Explore the space of possible inlining decisions

Continuously monitor user apps to drive optimizations

Group related object files for improving optimization search

Perform optimizations across libraries

Reuse past results for accelerating search

Fully automated build/run optimizer (ATOS)
ATOS function-by-function exploration

Optimization Space for HEVC decoder ARM/NEON [ref=REF]

- **Improvements** while hot functions are discovered
- **Current best perf tradeoff**
  - 8.44% speedup
  - 15.89% reduction

- **Best size tradeoff**
  - Used for cold functions

- **Reference (-O3)**

- **Legend**
  - frontier
  - opt cases

- **Axes**
  - speedup (higher is better)
  - size reduction (higher is better)
ATOS Status

- ATOS has been in production since 2 years
- ATOS FOSS-OUT is done (approval e/o February)
- ATOS to be made open source by March 2016 on GitHub
  - License GPL v2 and later
- STMicroelectronics initial gatekeeper
ATOS Delivery Script

ATOS surdriver script to install in compiler directory:

```bash
#!/bin/bash

CC_PATH="$\( cd \$\( \text{dirname} \$\{BASH\_SOURCE[0]\} \) \&\& pwd \)"


${CC\_PATH}/sh4-linux-uclibc-gcc $\{1+"$@"\} ${ATOS\_OPTS}
```
Thanks for your attention