Power Awareness through Selective Dynamically-Optimized Traces

Roni Rosner, Yoav Almog, Naftali Schwartz, Avi Mendelson, Micha Moffie, Ari Schmorak and Ronny Ronen

Intel Labs, Haifa

Compiler & Architecture Seminar at IBM Haifa
November 2003

http://www.intel.com/research/mrl/research/arch.htm
Agenda

PARROT =
Power-aware ARchitecture Running
Optimized Traces

- Concept and micro-architecture
- Performance and energy results
- Dynamic optimizations
Trends in a Nut-Shell

😊 Ideally, each process technology generation provides
- Reduction in transistor switching energy by 3X
- Reduction in transistor delay time by 1.5X
- Reduction in transistor switching power by 2X.

😊 More instructions per cycle, more cycles per second
→ gains performance but consumes power...

😊 More instructions per task (deep speculations)
- More leakage power
→ Higher energy consumption per task!

😊 Parrot tries to change the balance!
Parrot Concept and Microarchitecture

Microarchitecture to identify hot traces, optimize them, execute efficiently – awareness of both performance and power

- Extend the well known cold/hot (10 code / 90 execution) paradigm
  - frequently used code behaves differently
  - More regular, predictable
  - Coarser granularity (longer sequences of instructions)

- Parrot Principles
  - Reuse → trace-cache centric
  - Dynamic optimizations → more performance with less energy, hw-oriented
  - Focus → invest where it pays
  - Asymmetric decoupling → hybrid front-end, cold and hot execution pipelines
  - Transparency → immune to s/w compatibility, overcome legacy
Parrot 4-Phase Scheme

FETCH SELECTOR

BRANCH PREDICT
INST CACHE
TRACE PREDICT
TRACE CACHE

TRACE-based COLDEXEC
TRACE OPTIMIZER
TRACE CACHE
TRACE SELECT, FILTER & BUILD

IA32 based COLDEXEC
COLD EXEC
HOT EXEC
SYNCHRONIZED COMMIT

Cold
Trace Build
Hot
Optimize
Parrot in Context

- Software techniques:
  - JIT, profile-based compilation, binary-translators (Dynamo, Daisy, FX32!, IA32 Execution Layer for IA64), code morphing (Transmeta)

- Micro architectural techniques
  - Front end optimizations
    - Pentium® 4, UOP cache, RePlay (U of Illinois)
  - Asymmetric execution
    - Turbo-Scalar [Shen], DIVA [Austin]

- Our uniqueness
  - Filtering [PACT’01]
  - Smart trace selection [ICS’03]
  - Full IA32 context, including X87 floating-point model
  - Optimization for specialized execution [paper submitted]
  - Putting it all together [paper submitted]
μArchitecture Trade-Offs

PARROT Capabilities

Parrot
- IPC: +17%
- Energy: +4%
- P²/E: +33%

Base (4-wide OOO)

Parrot + Wide
- IPC: +45%
- Energy: +39%
- P²/E: +51%

Wide (8-wide OOO)
- IPC: +16%
- Energy: +70%
- P²/E: -21%

Conventional Widening

- WIDE FRONT-END
- WIDE EXECUTION

PARROT exhibits comparable performance and better power-awareness (P²/E)

Higher is better
Lower is better
Dynamic Optimizations

**Optimization Classes**
- **Trace selection**
  - Loop unrolling
  - Procedure inlining
- **Basic transformation**
  - Virtual renaming, SSA
  - FP-stack flattening
  - Handle partial regs
  - Color regs: \( LI, \ tmp, \ LO \)
  - Replace \( CTI \) by \( ASSERT \)
- **Generic optimizations**
  - Logic, arithmetic simplifications
  - Propagation of values, registers, conditions
  - Dead code elimination
- **Core-specific**
  - Uop fusion
  - \( SIMD \)ification
  - Semi-dynamic scheduling

**Benefits of Optimizations**
- **Less uops**
  - Higher performance
  - Reduced energy consumption
- **Reduced dependencies**
  - Higher ILP \( \rightarrow \) higher performance

**Efficiency of Optimizations**
- **Utilization**
  - High rate of \( \text{execute} / \text{optimize} \)
- **Sensitivity**
  - High benefits at low frequency of optimization
## Optimization Example [SysMark, MS Word]

<table>
<thead>
<tr>
<th>IA32 Instructions</th>
<th>Original PARROT uops</th>
<th>uops after basic-transformations</th>
<th>uops after optimizations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0. mov eax ← edi</td>
<td>0. eax ← mov (edi)</td>
<td>0. v_a(0) ← move (li edi)</td>
<td>0. v_c(1), v_c(3), lo_cx</td>
</tr>
<tr>
<td>1. dec edi</td>
<td>1. edi, eflags ← sub (edi, 1)</td>
<td>1. v_b(0), v_flags(0) ← sub (li edi, 1)</td>
<td>3Xload(li_ds, li_esi)</td>
</tr>
<tr>
<td>2. testl eax, eax</td>
<td>2. eflags ← and (eax, eax)</td>
<td>2. v Flags(1) ← and (v_a(0), v_a(0))</td>
<td>3Xsub(li edi, 1, li edi, 2, li edi, 3)</td>
</tr>
<tr>
<td>3. je +0x14</td>
<td>3. cond_jmp(e,eflags)</td>
<td>3. assert cond(vFlags.z(1), e, ntaken)</td>
<td>3. jmp (0x3004d8a5)</td>
</tr>
<tr>
<td>4. mov cx ← [esi]</td>
<td>4. cx ← load (ds, esi)</td>
<td>4. v_c(1) ← load(li_ds, li_esi)</td>
<td>3. lo_esi ← add(li_esi, 0x6)</td>
</tr>
<tr>
<td>5. cmp cx, 256</td>
<td>5. eflags ← sub (cx, 256)</td>
<td>5. v Flags(2) ← sub (v_c(1), 256)</td>
<td>4. lo_eflags ← sub&amp;assert(lo_cx, 256, nbe, ntaken)</td>
</tr>
<tr>
<td>6. ja 0x2c645e</td>
<td>6. cond_jmp(nbe,eflags)</td>
<td>6. assert cond(vFlags.z(2), nbe, ntaken)</td>
<td>5. and&amp;assert(v_b(3), v_b(3), e, ntaken)</td>
</tr>
<tr>
<td>7. inc esi</td>
<td>7. esi, eflags ← add (esi, 1)</td>
<td>7. v_b(1), v_flags(3) ← add (li, esi, 1)</td>
<td>6. sub&amp;assert(v_c(3), 256, nbe, ntaken)</td>
</tr>
<tr>
<td>8. inc esi</td>
<td>8. esi, eflags ← add (esi, 1)</td>
<td>8. v_b(2), v_flags(4) ← add (v_b(1), 1)</td>
<td>7. and&amp;assert(v_b(0), v_b(0), e, ntaken)</td>
</tr>
<tr>
<td>9. jmp -23</td>
<td>9. jmp(0x3004d8a5)</td>
<td>9. v_a(2) ← move (v_b(0))</td>
<td>8. sub&amp;assert(v_c(1), 256, nbe, ntaken)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9. and&amp;assert(li_EDI, li_EDI, e, ntaken)</td>
</tr>
<tr>
<td>10. mov eax ← edi</td>
<td>10. eax ← mov (edi)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11. dec edi</td>
<td>11. edi, eflags ← sub (edi, 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12. testl eax, eax</td>
<td>12. eflags ← and (eax, eax)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13. je +0x14</td>
<td>13. cond_jmp(e,eflags)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14. mov cx ← [esi]</td>
<td>14. cx ← load (ds, esi)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15. cmp cx, 256</td>
<td>15. eflags ← sub (cx, 256)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16. ja 0x2c645e</td>
<td>16. cond_jmp(nbe,eflags)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17. inc esi</td>
<td>17. esi, eflags ← add (esi, 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18. inc esi</td>
<td>18. esi, eflags ← add (esi, 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19. jmp -23</td>
<td>19. jmp(0x3004d8a5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20. mov eax ← edi</td>
<td>20. eax ← mov (edi)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21. dec edi</td>
<td>21. edi, eflags ← sub (edi, 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22. testl eax, eax</td>
<td>22. eflags ← and (eax, eax)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23. je +0x14</td>
<td>23. cond_jmp(e,eflags)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24. mov cx ← [esi]</td>
<td>24. cx ← load (ds, esi)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25. cmp cx, 256</td>
<td>25. eflags ← sub (cx, 256)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26. ja 0x2c645e</td>
<td>26. cond_jmp(nbe,eflags)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27. inc esi</td>
<td>27. esi, eflags ← add (esi, 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28. inc esi</td>
<td>28. esi, eflags ← add (esi, 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29. jmp -23</td>
<td>29. jmp(0x3004d8a5)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Optimization Example

Dependency graph of the trace after renaming & SSA, before optimizations
- 28 micro-ops
- tree-height 7

Dependency graph of the trace after optimizations
- 10 micro-ops
- Tree-height 2
Optimizer Quality

Optimization Impact

- Dependency reduction
- Uops reduction

Utilization of Optimized Traces

Higher is better

Parrot
Parrot + Wide
Optimizer Sensitivity: Impact of Filter’s Threshold on IPC, Energy

- Higher bars are better for overall performance, energy consumption
- Higher threshold \( j \) represent less sensitivity to optimizer cost, latency
- Tradeoff point: “knee” where smaller threshold has a low marginal contribution to performance \( \sim 512 \)

Legend
- \( N_j \) - Parrot, optimizer threshold is \( j \)
- \( W_j \) - Parrot+Wide, optimizer threshold is \( j \)
Optimizer Sensitivity: Impact of Filter’s Threshold on IPC, Energy

Relaxed design of optimizer is feasible!!!
Optimizations Breakdown: Generic vs. Core-Specific

IPC Contribution of Optimizations over Base

Higher is better

Energy Saving by Optimizations over Base

Higher is better
**Conclusions and Future**

**Major conclusions**
- Parrot-style \( \mu \) architecture presents a power-aware alternative to conventional design
- Parrot \( \mu \) arch shows low sensitivity to optimizer latency, greediness → feasibility of relaxed optimizer
- Core specific optimizations are central for hardware-based dynamic optimizations

**Interesting Future Directions**
- Microarchitecture
  - Improved selection, prediction
  - Specialized traces, value prediction
  - Multi-threading
- Architecture and software
  - Interaction with compilers, profiling, JIT
  - Quality of service
- Improved dynamic optimizations
Backup Slides
Front-End Characteristics

- Trace Cache Fetch-Coverage
  - Parrot
  - Parrot + Wide
  - Higher is better
  - Lower is better

- Normalized Mis-prediction Rate
  - Higher is better
  - Lower is better