Energy-Aware Computing

Lecture 11: Software-level techniques

Outline

- Compilers
  - Loop transformations
  - Instruction scheduling
- Remote task compilation and execution
- Dynamic compilation
- Application adaptation

What can compilers do?

- Exploit low-power instructions and processor modes
- Improve memory access “behaviour”
  - Locality of accesses
  - Number of accesses
  - Reduce memory area (embedded systems)
- Improve instruction scheduling
- Provide input for DVFS scheduler

Reducing memory energy

- Loop transformations
  - Interchange, skewing, unrolling, …
  - Tiling/blocking
  - Fusion/fission
- Data placement
  - Address assignment to variables
- Code compression
  - Loop buffers, reconfigurable instructions,…

Loop interchange

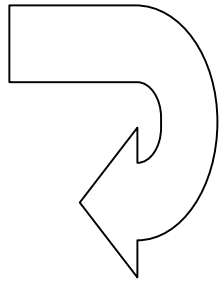
```
for i from 0 to 10
  for j from 0 to 20
   a[j,i] = b[j,i]
```

```
for j from 0 to 20
  for i from 0 to 10
   a[j,i] = b[j,i]
```

- Improves locality
  - Arrays accessed in sequence
- Usually improves cache performance and energy
  - But not always

Loop tiling/blocking

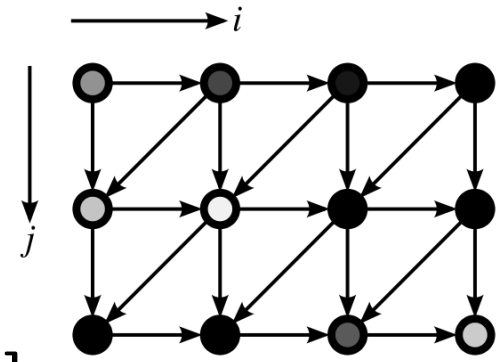
```
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        c[i] = c[i]+ a[i,j]*b[j];
```



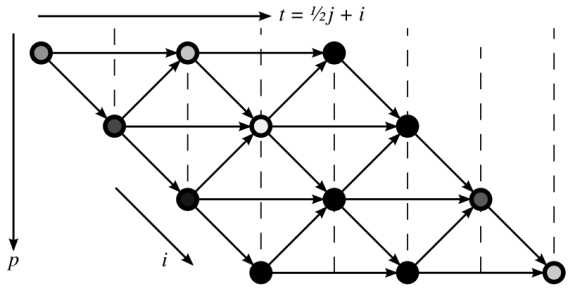
```
for (i=0; i<N; i+=2)
  for (j=0; j<N; j+=2)
    for (ii=i; ii<min(i+2,N); ii++)
      for (jj=j; jj<min(j+2,N); jj++)
        c[ii] =c[ii]+ a[ii,jj]*b[jj];
```

- Improves cache re-use
  - Partitions large arrays to fit in cache

Loop skewing




```
for i = 0 to n
   for j = 0 to i+2
        A[i,j] = A[i-1,j] + A[i, j-1]
```




```
for t = 0 to 2n+2
   for p = max(0,t-n) to min(t,floor(t/2)+1)
     A[t-p, p] = A[t-p-1, p] + A[t-p, p-1]
```

- Rearranges array accesses so that the only dependencies are between iterations of the outer loop

Power aware instruction scheduling

Parikh et al 2004

- Works at the basic block level
- Based on list scheduling
  - Build DAG of data dependencies
  - Information on instruction execution times and latency between dependent instructions
- Energy, 2 types:
  - Average energy for instruction execution
  - Circuit state cost: energy for scheduling one particular instruction after the other due to switching activity

Top-down approach

- Traverse DAG
- Candidate list of instructions that can be scheduled
- Select the candidate with max possible delay to the end of the basic block
  - Performance only algorithm
- A number of variations tried
  - Bottom up traversal
  - Energy x delay as cost function
  - Prioritize energy/delay for candidate selection

Instruction scheduling eval

- Best energy-only approach can reduce energy by up to 30% compared to speed scheduling
  - But speed drops by 6%
- Best results from combined energy-delay cost functions

Remote task compilation/execution

- Application partitioning between servers, wireless terminals
  - Or even remote compilation for rel. modern languages such as Java
- Compilation rule of thumb:
  - If compilation time > exec time, do it remotely
  - But communication energy can be important
- Not a one solution fits all. Depends on:
  - Code size, input size, communication energy (incl channel quality issues),…
  - See Chen et al for an analysis (*suggested reading*)

Dynamic compilation

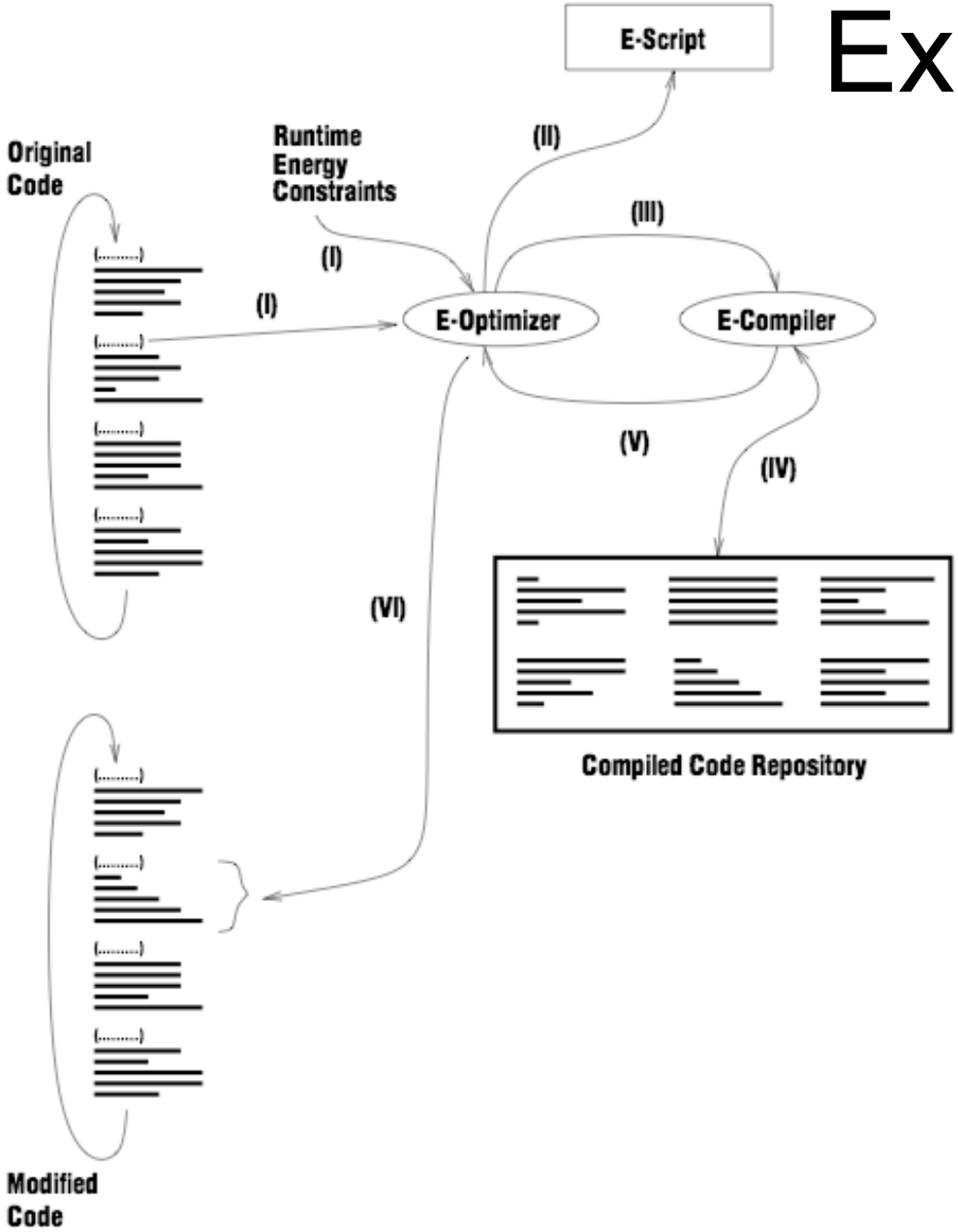
- Program is compiled and its execution is monitored
- If there are significant changes, the program may be recompiled
  - Transmeta's code morphing

Unnikrishnan et al 2002

Dynamic compilation

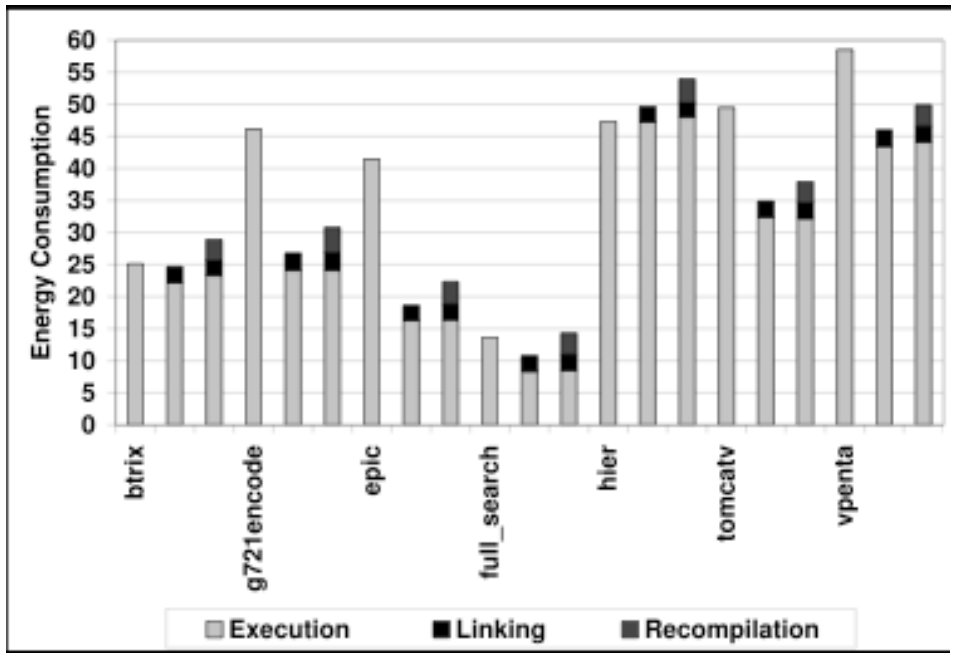
- Dynamic compilation using the Dyninst dynamic binary reconfiguration tool
  - Can instrument and modify a program during execution
- Application source augmented with sensitivity lists
  - For loops or functions
  - List declares which energy components it is sensitive to (cache, main memory, core)
  - How to identify energy-critical regions?

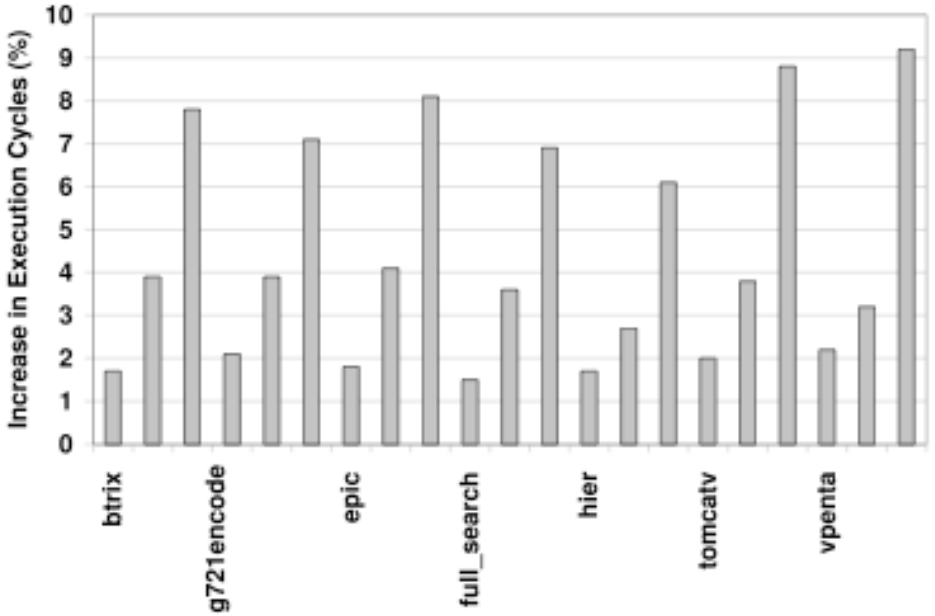
Execution model



- On entry to energy-critical region check for changes in constraints
- Replace original region with alternative
  - Pre-compiled in a repository
  - Or compile from source code using constraints

Evaluation





- Based on simulating a small, embedded processor

Application-level power management

Two main directions:

- Enable applications to adapt to runtime environment
  - Trade accuracy/fidelity for energy savings
  - E.g. (lossy) compression, multimedia, communication rates
- Develop interfaces for applications to provide hints to lower layers of software stack
  - Deadlines, I/O (disk) usage, …