Introduction to High Performance Computing for Scientists and Engineers

Chapter 2: Basic Optimization Techniques for Serial Code

Scalar Profiling

- *Profiling*: gathering data about program's use of resources
- Profiling helps determine which portions of program have greatest potential for reducing overall run time through code optimization
- Methods of profiling
  - code instrumentation (fine detail but high overhead)
  - periodic sampling (accuracy depends on length of run)
- Levels of profiling
  - individual lines
  - basic blocks (section of code with one entrance and one exit)
  - functions

Function Profiling

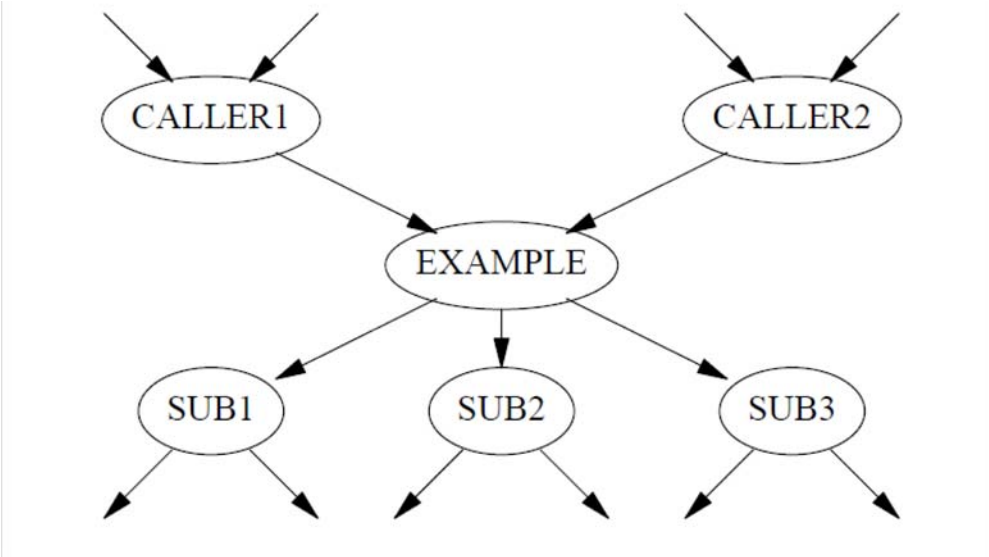
- Most widely used tool for function profiling, `gprof`, provides data on number of calls and time spent in each function in program

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
70.45	5.14	5.14	26074562	0.00	0.00	intersect
26.01	7.03	1.90	4000000	0.00	0.00	shade
3.72	7.30	0.27	100	2.71	73.03	calc_tile

- In addition to flat profile, can also produce call graph profile indicating relationship between calling and called functions
- Function profiling may be of little value for functions with many lines of code, in which case line-based profiling is more useful
- Can also do manual instrumentation by inserting calls to timer

gprof call graph profile

- Based on the call graph.
- One node per subroutine.
- Arcs join callers to callees:

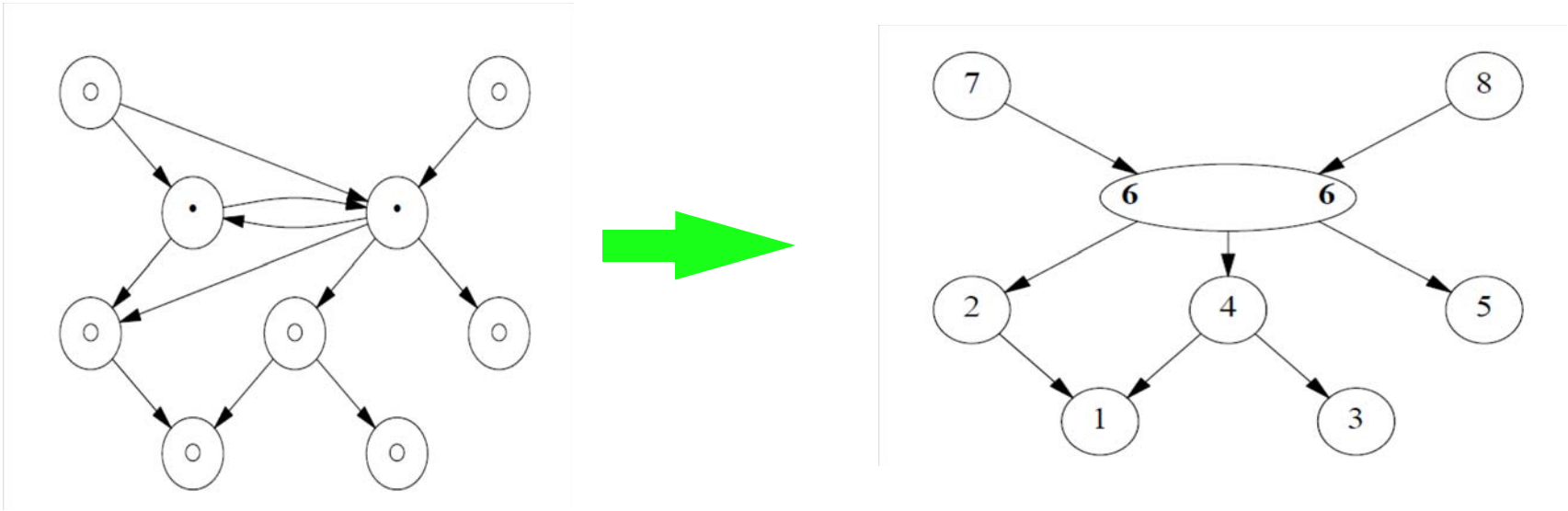


- Report is one node at a time and its relation to parents and chicren in the call graph:

index	%time	self	descendants	called/total called+self called/total	parents name children	index
		0.20	1.20	4/10	CALLER1	[7]
		0.30	1.80	6/10	CALLER2	[1]
[2]	41.5	0.50	3.00	10+4	EXAMPLE	[2]
		1.50	1.00	20/40	SUB1 <cycle1>	[4]
		0.00	0.50	1/5	SUB2	[9]
		0.00	0.00	0/5	SUB3	[11]

gprof call graph profile - Recursive routines

- Cycles in the call graph (strongly connected components) are collapsed into a single node.
- A directed graph is called **strongly connected** if there is a path from each vertex in the graph to every other vertex. In particular, this means paths in each direction; a path from **a** to **b** and also a path from **b** to **a**. (from wikipedia)
- The **strongly connected components** of a directed graph G are its *maximal* strongly connected subgraphs. (from wikipedia)



-

Hardware Performance Counters

- Most modern processors provide hardware performance counters that can report number of occurrences of various events
  - bus transactions (cache line transfers)
  - loads and stores
  - floating-point operations
  - mispredicted branches
  - pipeline stalls
  - instructions executed
- Appropriate metrics can be derived from these raw data, such as instructions per cycle or cache misses per load or store
- Data accessible through libraries such as `PAPI`

Compiler Optimization

- Most modern compilers offer aggressive levels of code optimization
- Occasionally, overly aggressive optimization can lead to incorrect results, so spot check your optimized results against unoptimized results (e.g., `-O3` vs. `-O0`)
- Many techniques for optimizing code amount to the programmer getting out of the compiler's way and letting it do its thing, for example by avoiding programming constructions that inhibit compiler optimizations

Basic Code Optimizations

- Avoid unnecessary work
- Avoid expensive operations, for example by *strength reduction*

  `x**2.0` vs. `x**2` vs. `x*x`

- Shrink working set (amount of memory touched), for example by using smaller word length whenever feasible
- Eliminate common subexpressions
- Avoid branches in loops
- Unroll loops
- Avoid function call overhead by inlining
- Avoid aliasing

Caveats

- Some data types, such as two-byte integers, may not be efficiently supported on some processors, so savings in memory may be offset by slower processing
- Rearrangement of arithmetic expressions, even if mathematically legitimate, may alter results (e.g., floating-point arithmetic is not associative)