Tempo Support in Programming Languages

Yu David Liu
SUNY Binghamton, NY 13902, USA
davidL@cs.binghamton.edu

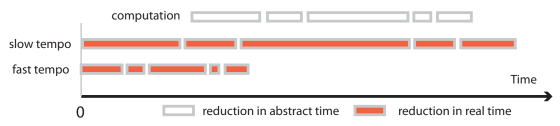
According to Wikipedia:

> "In musical terminology, **tempo** (Italian for time) is the speed or pace of a given piece. Tempo is a crucial element of any musical composition, as it can affect the mood and difficulty of a piece."

Just like musicians, software designers often think of themselves as artists and their programs as artistic compositions. If we carry out the analogy between music and programs further – notes as syntax, sound as semantics, symphony as program executions – the fundamental concept of tempo in music is naturally mapped to the "execution pace" of programs. What is eerily missing is the programming language foundations for tempo. In this short essay, we seek answers to three questions:

- *What* does "tempo" mean in programs?
- *How* can it be implemented in real-world machines?
- *Why* should we care about it?

Tempo intuitively can be thought of as *how fast* a program progresses. Imagine a computation operationally defined in five reduction steps, shown below as a sequence of gray transparent rectangles, the width of each representing its abstract reduction time, *i.e.* an abstract notion of CPU cycles:



When this computation is executed on a 300MHz CPU and a 1000MHz CPU respectively, the first will intuitively take longer time, in which case we say the first has a slower tempo than the second. A fundamental problem does arise from this example: the tempo here appears to be a characteristic of the external hardware environment – the CPU frequency – so how could it be considered as a *program semantic element*? It is our belief that, even though the *absolute tempo* of a program execution may be difficult to capture on the language level, the *relative tempo* of different program fragments within the same program can be effectively expressed and serves as a useful semantic feature. To set the analogy with music again, we believe that all layers of software design – semantics definition, optimizer specification, and programs themselves – should have a way of saying *lento*, *allegro*, and *accelerando*, so that different program fragments coordinate executions as composing a symphony.

The advances of hardware/software platforms in the last two decades significantly facilitate the implementation of program tempo adjustment. Virtually all CPUs being used today – from ARM Cortex on Droid smartphones, Intel Core 2 on laptops and desktops, to high-end clusters in data centers – are equipped with the ability of adjusting CPU frequencies (and voltages) on the fly, a feature commonly known as Dynamic Voltage and Frequency Scaling (DVFS).

To see why tempo can be a useful idea for programmers, consider a simple search from a large dictionary:

```
boolean find(String s) {
      accelerando for (long  i = 0; i< 200000; i++) {
            if (dictionary[i].equals(s) return true;
      }
      return false;
}
```

The **accelerando** keyword denotes that the loop execution "picks up" the tempo gradually: the execution starts at the lowest CPU frequency, and hopefully the search will "get lucky" when the 53rd entry – not the 53000th entry – of `dictionary` contains the string. If the search turns out not to be so lucky and the loop iteration reaches the 53000th entry, it becomes necessary to increase the CPU frequency. To use an analogy perhaps familiar with us researchers – work harder when the deadline looms! Because the CPU frequency is proportional to system energy consumption, the temp-variant execution here offers programmers an intuitive way to trade-off the quality of service (QoS) and energy consumption.

Tempo-variant executions may also be conducive for reducing concurrency bugs such as race conditions and deadlocks. The common nature of this category of bugs is that they are *situational*: whether they occur in an execution is non-deterministic, largely dependent upon the relative progress of different threads. Novel algorithms may be designed to allow different threads to execute on different tempi. For example, the occurrence of deadlocks may be reduced by increasing the tempo of thread A when it holds locks that may be requested by another thread B but in a different lock order, and decreasing the tempo of thread B at the same time.