RIT Department of Computer Science
MSc Thesis Pre-Proposal:
Process Cooperativity as a Feedback Metric in Functional Languages

Alexander Dean, Chair: Matthew Fluet

March 10, 2014

1 Problem Description

Runtime systems for functional languages have begun to utilize feedback mechanisms to influence their scheduling behaviour as the application proceeds. These feedback mechanisms rely on metrics by which to grade any alterations made to the runtime system. As the application's phase shifts, the feedback mechanism is tasked with modifying the runtime to reduce it's overhead and increase the application's efficiency.

For example, feedback mechanism could utilize the ratio of I/O to CPU bound processes within a set number of rounds to be the metric by which it grades it's efficiency. If it is too high the runtime should give each process less time to compute, so it can get through more processes in a round, thus reducing the drag the CPU bound processes have on the system. The inverse would have the opposite effect.

Cooperativity is another possible metric by which to grade a system. In biochemistry the term cooperativity is defined as the increase or decrease in the rate of interaction between a reactant and a protein as the reactant concentration increases. This definition translates well as an information theoretic definition as: the increase or decrease in the rate of interaction between a process and a communication mode as the number of processes increase.

I will attempt to answer the following questions:

- Are the abstractions of a process and communication channel conducive to quantifying cooperativity of a system? What generalizations can be made without obscuring important factors?
- What factors effect the cooperativity of a runtime system? Also, what mechanisms can be used to update the runtime system to positively improve cooperativity in a given phase?
- Is cooperativity a comparable metric for feedback scheduling?

2 Related Work

Scheduling based on feedback from the running system is not new [2]. There have even been multiple metrics explored, such as heap size [5], process locality [1], and interactivity [3]. In [4] they mention communication efficiency as a rational for their implementation of a runtime scheduler using process locality as the primary metric. However, this differs from capturing the cooperativity in that, while communication efficiency is important, it misses the quantity of communication which could indicate needed changes in scheduling quantum.

3 Motivation for Research

The motivation is twofold. Current feedback scheduling implementations fail to take process-cooperativity into account when applying feedback. A scheduler implementation and metric calculation would aid in future feedback schedulers. However, this hints at other missing metrics that could be useful for adapting to application phases. A closer evaluation of current scheduling techniques on common language primitives will allow for a constructive comparative analysis. This, as far as I'm aware, has not been done on the injected runtime level but rather the OS or Application levels with job schedulers.

4 Methodology

I have already begun to work on a compiler built with swappable scheduler's in mind. The language is a simple lambda and process calculi with synchronous swap channels so as to minimize the language characteristics which may mask the process-based metrics, such as differentiating between CPU, I/O, and Channel bound processes. This compiler will also allow me to directly test multiple metrics and scheduling techniques and visualize the system phases at any given point. I will attempt to do a comparative analysis of several popular scheduling techniques against a new algorithm I will need to design which uses cooperativity. This, in turn, will evaluate the effectiveness of the process and communication channel abstractions utilized in the language.

5 Potential Outcomes

A new tool for comparative analysis of some current runtime schedulers will be available as well as a set of high level language abstractions for future runtime scheduler testing.

Using this analysis I expect to also contribute an original scheduling algorithm which takes cooperativity of processes into account within it's feedback mechanism. This will also require an examination of fairness guarantees of both the communication mode and process selection.

References

[1] Kurt Debattista, Kevin Vella, and Joseph Cordina. Cache-affinity scheduling for fine grain multithreading. *Communicating Process Architectures*, 2002:135–146, 2002.

[2] Richard D Dietz, Thomas L Casavant, Mark S Andersland, Terry A Braun, and Todd E Scheetz. The use of feedback in scheduling parallel computations. In *Parallel Algorithms/Architecture Synthesis, 1997. Proceedings., Second Aizu International Symposium*, pages 124–132. IEEE, 1997.

[3] John H Reppy. Concurrent ml: Design, application and semantics. In *Functional Programming, Concurrency, Simulation and Automated Reasoning*, pages 165–198. Springer, 1993.

[4] Carl G Ritson, Adam T Sampson, and Frederick RM Barnes. Multicore scheduling for lightweight communicating processes. *Science of Computer Programming*, 77(6):727–740, 2012.

[5] David R White, Jeremy Singer, Jonathan M Aitken, and David Matthews. Automated heap sizing in the poly/ml runtime. *Trends in Functional Programming*, 2012.