

# Taming Irregularity and Dynamics at Exascale with Unspeculative Compilation and Predicative Runtime Accommodation

Xiaoming Li, Guang R. Gao (xli, ggao@udel.edu)

University of Delaware

Dynamic and irregular applications are hard to optimize because just like the name implies, system software are uncertain about their behavior and performance. How do we optimize something that we have only limited knowledge? An instinctive response might be “let’s predict”. Knowledge, even partial knowledge, about their behavior will enable better optimization strategies. However, we doubt the root of irregularity can be tamed. Because if we have a simple and concise predictor for an irregular application such as the many physical simulations in DOE applications, we simply don’t need to simulate in the first place.

A far more common approach is to chase after the irregularity and dynamics, i.e., constantly adapting the system software behavior, i.e., resource allocation, to the dynamic and irregular computation/communication patterns. In other words, this optimization strategy is determined from the observation of behavior at runtime. At either compilation time or runtime, extensive profiling is employed, and the statistics from profile determine the optimization strategy to be used in the next step. Essentially, system software speculates. Speculation-based approaches work very well for applications with regular computation and communication patterns because in those applications history is indeed a good reference for the future. By measuring the past, system software can more or less accurately gauge the profit-risk-ratio for a specific transformation, and in this case the selection is well justified.

The speculative approaches, however, will not work for irregular and dynamic applications and will not scale up to extreme parallel computers. First of all, the assumption “*history=future*” is fundamentally questionable for dynamic and irregular applications. For example, in the fluid dynamic simulation, the computation at one time step can behave very differently from that in the previous step even in the same physical unit because parameters such as the density of solid particles and the velocity of fluid can radically change in between. Speculating the future optimization strategy from the observation of now is at very least dangerous for this type of applications. Moreover, speculative optimization, almost without exception, requires intimate observations and modification of program behavior. The issue is, the associated data gathering and transfer will become relatively more expensive as problem size increases because data is more spread out and more data need to be gathered and transferred. When done at extreme scale, the overhead of that intimacy will quickly diminish its benefit.

Let’s take a deeper look at exactly what are the effects of the uncertainty on compilation and runtime support for programs. From the 10,000-foot point-of-view, the effect is averaging optimization and wasteful resource allocation. Just because a compiler cannot expect what a piece of code will process, the code must be accommodated, by compiler or runtime support, for all possible cases, and therefore, optimization decisions are averaged out for all possibilities. Clearly, such optimization strategy is a compromise for all and is not best for any. Furthermore, since the different possible workload handled by a piece of code require different amount and type of resources, to prepare for all, the union of all the required resources is allocated. Reasonable on surface, dearly wasteful on root. At any moment, the resource allocated is more than what is needed to process the workload at hand. The resource waste is particularly harmful for extreme-scale computing because the performance for an extreme-scale program is usually determined by throughput, i.e., how many threads can concurrently execute useful work. The resource waste will hurt concurrency, the main source of extreme-scale performance.

The observation of averaging optimization and wasteful resource allocation points to an interesting contrarian question: instead of chasing after the irregularity, can compilers and runtime support stop speculating but proactively reduce uncertainty in irregular applications? In other words, is it possible to proactively avoid the guessing in the selection of optimization strategy and choose to be more preciser about resource allocation, even doing so might betray traditional performance metrics? The answer to that contrarian question motivates this proposed approach. Essentially, the novelty of the proposed unspeculative compiler and runtime software paradigm is the counter-intuitive lowering the uncertainty of irregular programs and translate that higher confidence on program behavior into better overall performance, even though our uncertainty-averting optimization strategy might seemly generate sub-optimal code compared with traditional approaches. In simpler words, the surer can be faster than the hasty.

As the project title summarizes, we focus on the part of system software that directly interact with programs, which includes compiler and runtime environment. All the component collaborate to find, develop and benefit from the orders in the seemingly chaotic behavior of irregular applications. The research goal can be divided into two tasks: (1) Find the opportunities, existing or revealed after program transformation, in dynamic and irregular applications that lead to better behavior predictability; and (2) use the better behavior modeling to make a better optimization strategy. We plan a synergy of three research thrusts to accomplish the two tasks. The first research thrust explicitly defines and quantifies the concept of uncertainty in Exascale applications. The second research thrust builds program transformations specifically for lowering the uncertainty of the program behavior. And the third research thrust is the coordination of all compilation transformation and runtime adaptations to purposefully minimize the uncertainty of a program and translate the reduced uncertainty into better overall performance.

**Qualities of the Approach** An assessment of the approach along the suggested dimensions:

- **Challenges Addressed:** Irregularity in Exascale applications, and its impact on resource management at compilation time and runtime.
- **Maturity:** Compilation techniques have been developed in the context of GPU-like architectures. A small scale Fresh Breeze simulator is implemented, as well as a variety of benchmarks.
- **Uniqueness:** The efficient management of resources, in particular those shared resources in Exascale systems is one of the most daunting challenges in system software. The proposed approach is a head-on attack on this unique challenge.
- **Novelty:** Instead of chasing irregularity, reducing irregularity and translating the lower irregularity into better resource management, and consequently into better performance is a contrarian idea and is unique.
- **Applicability:** Applicable over general purpose applications on Exascale systems, but will demonstrate the largest benefit for application with irregular computation and resource utilization patterns.
- **Effort:** Scale up the implemented resource-centric compiler to thousands of cores. Augment the Fresh Breeze simulator with resource-guided dynamic configuration.