Plastic Skeletons

Paul Metzger, Murray Cole,[*]
Christian Fensch[†][1][2]

[*] *University of Edinburgh, Informatics Forum, 10 Crichton Street, Edinburgh, EH8 9AB, United Kingdom*
[†] *Heriot-Watt University, Riccarton, Edinburgh, EH14 4AS, United Kingdom*

ABSTRACT

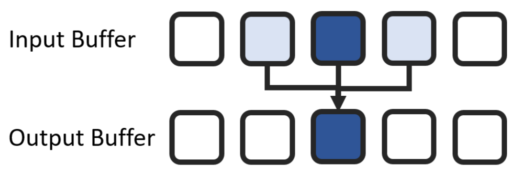
Increasing heterogeneity of current systems as well as unpredictable resource availability and competition make it desirable for modern applications to adapt to disruptions at runtime. We call this dynamic behaviour plasticity.
Current runtime systems such as OpenMP are not aware of disruptions and cannot react to them. Popular parallel programming language models such as pthreads make some plastic behaviour such as changing thread counts even impossible as this would change the program semantics.
We propose to hide the program logic for plasticity behind high level programming language constructs in a library and runtime system. This runtime system can make globally optimal decisions on how applications should change their execution strategies in response to disruptions. Our preliminary results show that significant speedups can be gained with plasticity.

KEYWORDS: Runtime Systems; Scheduling; Algorithmic Skeletons

1 Motivation & Background

Increasing heterogeneity of current systems as well as unpredictable resource availability and competition make it desirable for modern applications to adapt to disruptions at runtime. We call this dynamic behaviour **plasticity**.
Current applications do not handle disruptions well for three reasons. Firstly, applications and runtime systems are not aware of disruptions that occur at runtime. Typical applications execute with the implicit assumption that properties of their execution context such as resource availability do not change. Secondly, there is no mechanism to change execution parameters such as the number of threads that an application uses and their placement at runtime. Programming models such as pthreads make some plastic behaviour such as changing the number of threads even impossible as this would change the program semantics. Thirdly, current runtime systems are not multiprogram aware and cannot make globally optimal decisions. Our preliminary research results show that plastic runtime systems can achieve significant speed ups in multiprogram scenarios.

[1]E-mail: {s1576303,mic}@inf.ed.ac.uk, c.fensch@hw.ac.uk
[2]This work was supported by grant EP/L01503X/1 for the University of Edinburgh School of Informatics Centre for Doctoral Training in Pervasive Parallelism (http://pervasiveparallelism.inf.ed.ac.uk/) from the UK Engineering and Physical Sciences Research Council (EPSRC).



(a) 1D stencil computation

```
stencil(input_buffer,
        output_buffer,
        user_function)
```

(b) Stencil skeleton function call

Figure 1: Illustration of a stencil computation and a call to a stencil skeleton.

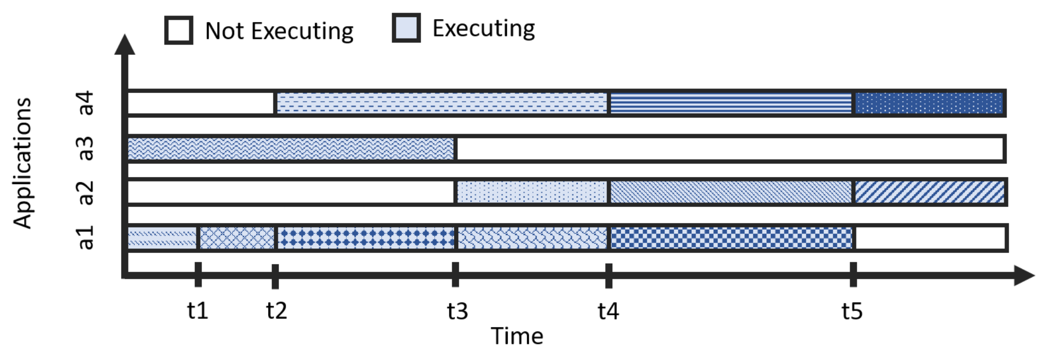


Figure 2: Illustrates the execution of plastic applications a1 to a4. Different patterns indicate execution strategies. Strategies are changed in response to disruptions that occur at t1 to t5.

We hypothesise that the program logic for plasticity can be hidden behind high level programming language constructs, so called algorithmic skeletons, so that application developers can be oblivious to it.
Section 1.1 introduces algorithmic skeletons. Section 2 discusses plasticity. Section 2.1 and 2.2 discuss disruptions and execution strategies. Section 3 concludes this abstract.

1.1 Algorithmic Skeletons

Algorithmic skeletons are implementations of design patterns in parallel programs [Col04]. The parallel execution is implemented by a skeleton library or compiler. The programmers' task is to implement domain specific code that is then passed to the skeleton.
Stencil computations are one example for parallel design patterns. Figure 1a illustrates a stencil computation with a 1D in- and output buffer. The computation updates each element. The new value depends on the old value of the element and the values of the elements in its neighbourhood. Elements are depicted by squares. The size, shape and dimensionality of the neighbourhood varies across stencil computations. Stencil computations can be iterative. The output of an iteration becomes the input of the next iteration.
Figure 1b shows a call to a skeleton function that implements a stencil computation. The programmer has to pass an in- and output buffer as well as a user defined function to the skeleton. The user defined function computes the new values of the elements. One call to the skeleton function can perform multiple iterations.

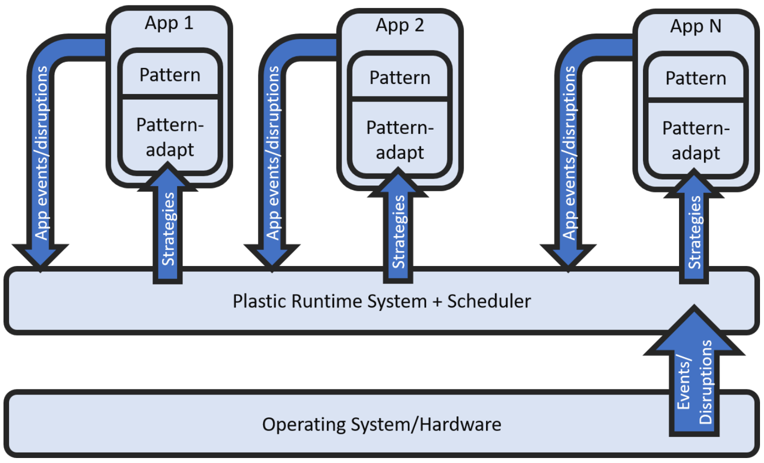


Figure 3: Plastic runtime system and plastic applications.

2 Plasticity

Plastic applications change their **execution strategies** in response to **disruptions** at runtime. The applications do this to optimise for metrics such as joint performance of the applications or energy consumption. Figure 2 illustrates the execution of four plastic applications. The applications change their execution strategies over time. Strategies encompass resource utilisation, data layouts and algorithms and are discussed in section 2.2.
Figure 3 shows how we envision a plastic runtime system and plastic applications. Applications send information about their computations and runtime events to the plastic runtime system. Information about the system such as hardware performance counters are provided by the operating system and the hardware to the plastic runtime system. The runtime system makes then globally optimal decisions on execution strategies that the applications should adopt. Execution strategies include the choice of algorithm, resource utilisation and the data structures that are used. Section 2.2 discusses execution strategies.

2.1 Typical Disruptions

Disruptions occur at runtime and have a negative performance impact or provide opportunities for performance improvements. Plasticity can mitigate their negative effects or exploit these opportunities. We identified three different classes of disruptions. These are described below.

Changing Resource Properties Resource properties such as availability can change unpredictably at runtime. For example, when other applications start and stop to use them or when the system deactivates resources to save energy.

Changing Data Properties Properties of in- and output data as well as internally used data can change at runtime. For example, the rate at which new input data arrives can change dynamically in streaming applications.

Changing Program Properties Properties of applications can change at runtime. For example, the required precision of output can change dynamically.

2.2 Dimensions of Plastic Execution Strategies

Applications change their execution strategies in response to disruptions. For example, to improve their performance. Execution strategies determine:

Hardware Utilisation Applications can change the way in which they use resources in response to disruptions. For example, plastic applications can dynamically change the number of threads that they use to avoid over- or undersubscription.

Algorithms Sets of algorithmic choices with different performance characteristics exist for many problems. Plastic applications use the most suitable algorithm based on hardware architecture, problem size, etc.. Previous work has shown that significant speedups can be achieved this way. [ACW+09].

Data Structures Data structures have to be changed when computations are migrated from one architecture to another. For example, previous work has shown that different memory layouts have to be used for GPUs and CPUs to make best use of their memory systems [NSC+10].

3 Conclusion & Future Work

This abstract presents a novel solution to limitations of current runtime systems and the outcomes will help to improve the performance and ease of programming of future systems. Our preliminary results show that applications can gain significant speedups through plasticity. We focus on stencil computations and NUMA systems currently and want to expand our work to the more general task farm and pipeline patterns in the near future.

References

[ACW+09] Jason Ansel, Cy Chan, Yee Lok Wong, Marek Olszewski, Qin Zhao, Alan Edelman, and Saman Amarasinghe. *PetaBricks: a language and compiler for algorithmic choice*, volume 44. ACM, 2009.

[Col04] Murray Cole. Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming. *Parallel computing*, 30(3):389–406, 2004.

[NSC+10] Anthony Nguyen, Nadathur Satish, Jatin Chhugani, Changkyu Kim, and Pradeep Dubey. 3.5-d blocking optimization for stencil computations on modern cpus and gpus. In *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*, pages 1–13. IEEE, 2010.