Abstract. Heterogeneous parallel systems incorporate diverse models of parallelism within a single machine or across machines and are better suited for diverse applications [28, 43, 30]. These systems are already pervasive in industrial and academic settings and offer a wealth of underutilized resources for achieving high performance. Unfortunately, heterogeneity complicates software development. We believe that compilers can and should assist in handling this complexity. We identify four goals for extending compilers to manage heterogeneity: exploiting available resources, targeting changing resources, adjusting optimization to suit a target, and allowing programming models and languages to evolve. These goals do not require changes to the individual pieces of existing compilers so much as a restructuring of a compiler's software architecture to increase its flexibility. We examine six important parallelizing compilers to identify both existing solutions and where new technology is needed.

1 Introduction

Heterogeneous processing

Current parallel machines implement a single homogeneous model of parallelism. As long as this model matches the parallelism inherent in an application, the machines perform well. Unfortunately, large programs tend to use several models of parallelism. By incorporating multiple models of parallelism within one machine (e.g., Meiko CS-2, IBM SP-2, and IUA [42]) or across machines, creating a virtual machine (e.g., PVM [38], p4 [9], and MPI [28]), heterogeneous systems provide consistent high performance.

Heterogeneous processing [36, 40, 41, 24, 18] is the well-orchestrated use of heterogeneous hardware to execute a single application [24]. When an application encompasses subtasks that employ different models of parallelism, the
application may benefit from using disparate hardware architectures that match
the inherent parallelism of each subtask. For example, Klietz et al. describe
their experience executing a single application, a simulation of mixing by turbu-
 lent convection, across four machines (CM-5, Cray-2, CM-200, and an SGI)[25].
The four machines form a single virtual machine, and the authors leverage the
strengths of each machine for different tasks to achieve high performance.

Their experience illustrates that although heterogeneous processing offers im-
proved performance, it increases the complexity of software development. The
complexity arises from three important features of heterogeneity: variety, variabil-
ity, and high performance. First, heterogeneous systems consist of a variety
of hardware. For an application to take advantage of heterogeneity, it must be par-
titioned into subtasks, and each subtask mapped to a processor with a matching
model of parallelism. Variety also opens up opportunities to trade local perfor-
mance for overall performance. Second, virtual heterogeneous systems experience
variability as their makeup changes from site to site or day to day or based on
load. This variability of hardware resources requires rapid adaptation of pro-
grams to new configurations at compile and run time. Furthermore, variability
deters programmers from using machine specific code (or languages) to improve
performance. Third, heterogeneous systems can achieve high performance. If the
execution time of a program does not matter, it could run on a homogeneous
processor with less trouble. The demand for high performance precludes simple
solutions such as adding layers of abstraction that obscure heterogeneity.

Compilers for heterogeneous systems

Developing software for heterogeneous systems would be overwhelming if each
application needed to handle the complexity caused by variety, variability, and
high performance. In Kleitz et al., they hand-parallelized each task specifically
for its target machine in that machine’s unique language dialect. If the hardware
configuration changes, they must rewrite parts of the program. Instead of man-
ually modifying programs, the variability of heterogeneous systems should be
automatically handled at least in part by a compiler. With certain modifications
to their software architecture, compilers can use transformations1 to adjust a
program to execute efficiently on a heterogeneous system.

Extending compilers to manage heterogeneity must address four goals: ex-
 ploiting available resources, targeting changing resources, adjusting optimiza-
tions to suit a target, and allowing programming models and languages to evolve.
(Sect. 3 explains why these goals are important.) Meeting these goals does not
require changes to the individual pieces of a compiler so much as a restructuring
of the compiler’s software architecture to increase its flexibility. Current com-
 pilers, including retargetable compilers, tightly couple the compiler with both
the source language and the target machine. This coupling is natural for homo-
genous machines, where a single compilation can only target a single machine.
However, this coupling limits the compiler’s flexibility in dealing with diversity in
targets, optimization strategies, and source languages. Heterogeneity is the first
compiler application that requires and therefore justifies this level of flexibility.

1 For brevity, “transformations” refers to both optimizations and transformations.