Leveraging Transparent Data Distribution in OpenMP via User-Level Dynamic Page Migration*

Dimitrios S. Nikolopoulos[1], Theodore S. Papatheodorou[1], Constantine D. Polychronopoulos[2], Jesús Labarta[3], and Eduard Ayguadé[3]

[1] Department of Computer Engineering and Informatics
University of Patras, Greece
{dsn,tsp}@hpclab.ceid.upatras.gr
[2] Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
cdp@csrd.uiuc.edu
[3] Department of Computer Architecture
Technical University of Catalonia, Spain
{jesus,eduard}@ac.upc.es

Abstract. This paper describes transparent mechanisms for emulating some of the data distribution facilities offered by traditional data-parallel programming models, such as High Performance Fortran, in OpenMP. The vehicle for implementing these facilities in OpenMP without modifying the programming model or exporting data distribution details to the programmer is user-level dynamic page migration [9,10]. We have implemented a runtime system called *UPMlib*, which allows the compiler to inject into the application a smart user-level page migration engine. The page migration engine improves transparently the locality of memory references at the page level on behalf of the application. This engine can accurately and timely establish effective initial page placement schemes for OpenMP programs. Furthermore, it incorporates mechanisms for tuning page placement across phase changes in the application communication pattern. The effectiveness of page migration in these cases depends heavily on the overhead of page movements, the duration of phases in the application code and architectural characteristics. In general, dynamic page migration between phases is effective if the duration of a phase is long enough to amortize the cost of page movements.

* This work was supported by the E.C. through the TMR Contract No. ERBFMGECT-950062 and in part through the IV Framework (ESPRIT Programme, Project No. 21907, NANOS), the Greek Secretariat of Research and Technology (Contract No. E.D.-99-566) and the Spanish Ministry of Education through projects No. TIC98-511 and TIC97-1445CE. The experiments were conducted with resources provided by the European Center for Parallelism of Barcelona (CEPBA).

M. Valero et al. (Eds.): ISHPC 2000, LNCS 1940, pp. 415–427, 2000.

1 Introduction

One of the most important problems that programming models based on the shared-memory communication abstraction are facing on distributed shared-memory multiprocessors is poor data locality [3,4]. The non-uniform memory access latency of scalable shared-memory multiprocessors necessitates the alignment of threads and data of a parallel program, so that the rate of remote memory accesses is minimized. Plain shared-memory programming models hide the details of data distribution from the programmer and rely on the operating system for laying out the data in a locality-aware manner. Although this approach contributes to the simplicity of the programming model, it also jeopardizes performance, if the page placement strategy employed by the operating system does not match the memory reference pattern of the application. Increasing the rate of remote memory accesses implies an increase of memory latency by a factor of three to five and may easily become the main bottleneck towards performance scaling.

OpenMP has become the de-facto standard for programming shared-memory multiprocessors and is already widely adopted in the industry and the academia as a simple and portable parallel programming interface [11]. Unfortunately, in several case studies with industrial codes OpenMP has exhibited performance inferior to that of message-passing and data parallel paradigms such as MPI and HPF, primarily due to the inability of the programming model to control data distribution [1,12]. OpenMP provides no means to the programmer for distributing data among processors. Although automatic page placement schemes at the operating system level, such as first-touch and round-robin, are often sufficient for achieving acceptable data locality, explicit placement of data is frequently needed to sustain efficiency on large-scale systems [4].

The natural means to surmount the problem of data placement on distributed shared-memory multiprocessors is data distribution directives [2]. Indeed, vendors of scalable shared-memory systems are already providing the programmers with platform-specific data distribution facilities and the introduction of such facilities in the OpenMP programming interface is proposed by several vendors. Offering data distribution directives similar to the ones offered by High-performance Fortran (HPF) [7] in shared-memory programming models has two fundamental shortcomings. First, data distribution directives are inherently platform-dependent and thus hard to standardize and incorporate seamlessly in shared-memory programming models like OpenMP. OpenMP seeks for portable parallel programming across a wide range of architectures. Second, data distribution is subtle for programmers and compromises the simplicity of OpenMP. The OpenMP programming model is designed to enable straightforward parallelization of sequential codes, without exporting architectural details to the programmer. Data distribution contradicts this design goal.

Dynamic page migration [14] is an operating system mechanism for tuning page placement on distributed shared memory multiprocessors, based on the observed memory reference traces of each program at runtime. The operating system uses per-node, per-page hardware counters, to identify the node of the