Adaptively caching data structures in bioinformatic applications with ReclamaMem

Luis Barrueco, FaMAF, UNC, Córdoba, Argentina.
Daniel Gutson, FuDePAN, Córdoba, Argentina.

Introduction

In the domain of bioinformatics, solving a particular problem computationally when dealing with molecular structures often requires access to a vast processing power and memory to manipulate the huge data volumes that they involve. When problems of this kind are being solved by non-profit organizations, every resource available must be fully used, being physical memory an important example of such a resource.

Bioinformatic algorithms often need a cache of partial computations, so that intermediate results can be reused when they are needed. This situation forces the researcher -frequently not a full time programmer- to consider the memory requirements of the data set and underlying algorithms, and the constraints imposed by memory availability in the computer system. Even worse, hardware resources are usually shared among different users and processes, hence memory availability varies in time and it's impossible to know beforehand. An important issue arises when the system runs out of main memory and uses secondary, high-latency memory to compensate. This turns into a performance penalty, and as we intend to show for this particular set of problems, most of the time it's faster to recalculate some results than to retrieve them from the swapped out portion of the cache.

The goal of this work will be to develop a caching library that will show two distinctive features: It will be able to recalculate results evicted from main memory, and it will make informed decisions regarding the expansion or shrinking of its own capacity. The latter will require access to specific knowledge from the memory subsystem in the underlying operating system.

Memory access speed

Swap memory exists in modern operating systems to keep programs working when the computer runs out of physical memory. However, for some high performance computing (HPC) applications, the speed degradation that it produces it's sometimes unacceptable. Figure 1 illustrates a typical scenario of the delay imposed by a "page fault" on modern hardware.

Figure 1: Latency of the retrieval of swapped memory

$$Page\ fault\ delay = \begin{cases} Spin-up\ time \\ Seek\ time \\ Rotational\ delay \\ Transfer\ time \\ Page\ replacement\ algorithm \end{cases}$$

ReclamaMem implementation

We will develop this library using the primitives provided by the Linux kernel, and we will use FuDePAN's Parallel Clusterer of Protein Backbones [1] over FuD [2] as the case study for our proposed solution. This algorithm partitions a set of protein backbones according to their similarity, by computing a distance function between related atoms in every pair of proteins in the set. Our caching layer should prove useful for avoiding multiple computations of the same distances, given the iterative nature of the solution by successive approximations.

It is worth noting that FuD distributes its workload among heterogeneous computing nodes, so the programmer often lacks the information about available memory in every node beforehand, and thus it is convenient to cache greedily every piece of data. [3]

Design alternatives

There is a crucial design decision that we will have to make to get to an actual implementation of this project, which is the feedback mechanism with the underlying memory subsystem in the Linux kernel. We have two main alternatives:

- mincore(): This is a user-space primitive provided by the Linux API, which allows the developer to know whether a set of virtual memory contiguous pages are resident in physical memory. Though, it has a downside: the programmer is responsible for dismissing those pages evicted to swap by the Virtual Memory Manager.

- Transcendent memory [4]: This is a Linux kernel extension that implements a kernel-space ephemeral memory abstraction. When Linux is short on physical memory, it can discard tmem backed pages without taking them to secondary storage.

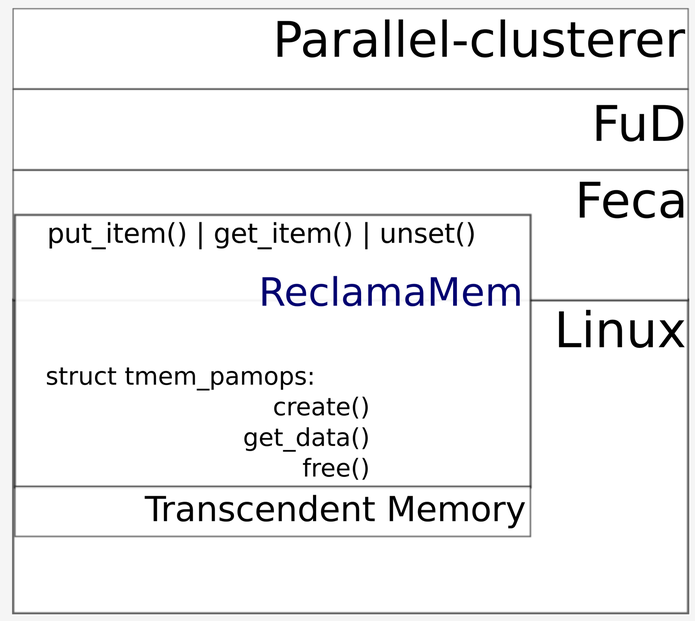
In any case it will be necessary to provide to the applications using our library with mechanisms to recalculate data discarded from memory.

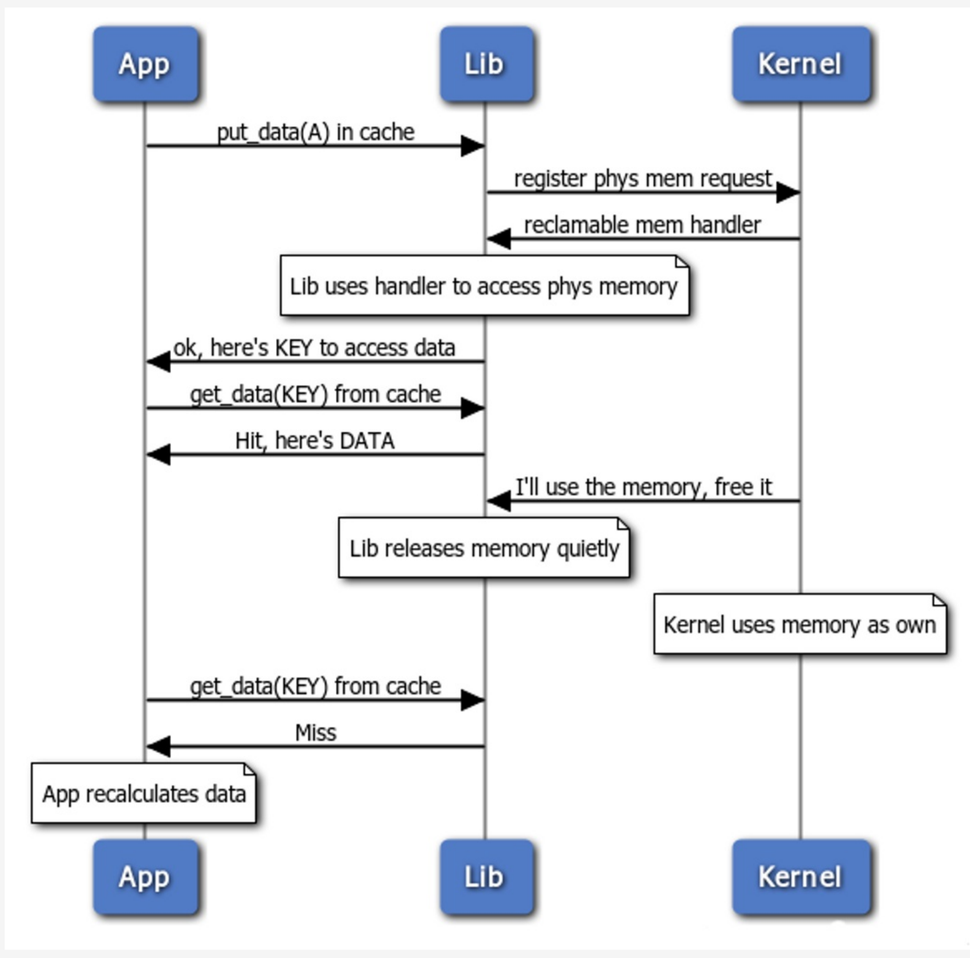
Performance analysis

We will perform an in-depth analysis of the results obtained from this work to get the actual impact of the effectiveness of our approach. For this purpose we will use benchmarking and profiling tools such as TAU ("Tuning and Analysis Utilities"), a toolkit for performance analysis of parallel programs, and PAPI ("Performance Application Programming Interface"), an API that provides access to the microprocessor performance counters.

Figure 2: Design of the proposed solution

Cache Size	Estimate	Bold	Swapping
		Cautious	Under-utilised memory
	Adapt at runtime	The PFRA evicts pages as necessary	





References

[1] A Framework for Small Distributed Projects and a Protein Clusterer Application, Guillermo Biset, UNRC (2009).

[2] FuDePAN Distributing Platform. http://code.google.com/p/fud/

[3] Operating System Support for General-Purpose Memory-Adaptive Applications, Neal H. Walfield, Johns Hopkins University (2009).

[4] Transcendent Memory and Linux, Dan Magenheimer, Oracle Corp (2010). http://oss.oracle.com/projects/tmem/