A Multimedia Interactive Environment Using Program Archetypes: Divide-and-Conquer

Paul Ainsworth
Svetlana Kryukova

Mail Stop 256-80
Department of Computer Science
California Institute of Technology
Pasadena, CA 91125
email: ainswrth@csvax.caltech.edu
kryukova@cco.caltech.edu

Abstract

As networks and distributed systems that can exploit parallel computing become more widespread, the need for ways to teach parallel programming effectively grows as well. Even though many colleges and universities provide courses on parallel programming [1], most of those courses are reserved for graduate students and advanced undergraduates. There is a demand for ways to teach fundamental parallel programming concepts to people with just a working knowledge of programming. By using the idea of a software archetype, and providing a learning environment that teaches both concept and coding, we hope to satisfy this need. This paper presents an overview of the multimedia approach we took in teaching parallel programming and offers Divide-and-Conquer as an example of its use.

Introduction

Parallel computers offer the potential for affordable high performance computation; this potential is limited by inadequate methods and tools for parallel programming and by an insufficient number of practical textbooks, courses and teaching tools. Parallel networks of PCs are being sold at prices that bring them well within the financial means of state and community colleges, as well as small businesses. Many developing programmers now have access to parallel technology, and therefore more effort should be donated to give them the insights that will allow them to understand and use this technology. With this in mind we sought to create learning tools that will provide people with opportunities to learn the fundamental principles of parallel programming, and thus allow them to comprehend and exploit these principles.

The guiding principle that shaped the design of our endeavor was the idea that the ultimate goal of the group is to teach, and that the intrinsic learning value of our modules should always be our paramount concern. With this idea in mind, we created an instructional package to teach parallel archetypes. This package was designed to teach parallel programming to people with varying amounts of computer science experience, with the focus on a person who is familiar with the basics of structured sequential programming (the equivalent of an introductory programming class at a college or university).

Many classes that teach proper algorithm design try to teach software principles to their students by using one of two methods:

1) The student is given a problem to write an algorithm to solve, and after the student has written the algorithm, the teacher presents the correct, efficient algorithm to the student.
2) The student is presented with several efficient and correct algorithms, and then asked to write a correct and efficient algorithm to solve a new problem.

Though both of these approaches have merit, we feel that they do not emphasize the process that is involved in writing good code. The student is presented with many correct algorithms, and is then expected to know how to write them. Expecting a student to develop good coding technique on their own after seeing correct algorithms is like giving a person a list of fifty example functions and their derivatives, and then asking him to understand all the derivative rules of calculus. In order to teach a student how to write correct and efficient algorithms, we need to present the student with a methodology for finding the right answer instead of showing the student the right answer and then hoping the student will somehow determine the method on their own. We want our package to do more than teach the student how to write parallel algorithms; we want to teach the student how to write correct and efficient parallel algorithms, as well as prove the algorithms correct, and discuss their efficiency. The way we present methods of solving problems in computer science using efficient algorithms is by teaching the student about the various software archetypes using interactive multimedia.

What are archetypes?

An archetype [2] is a general framework for solving a problem in computer science. There are many different archetypes, each one representing a unique strategy for developing an algorithm when faced with a problem. Problems that can be solved using a specific archetype conform to a recognizable pattern, and the majority of computer science problems can be solved using one archetype from a relatively small set.

An archetype presents a solution strategy in a general format that is language-independent and problem-independent. Each archetype presents the strategy it represents in terms of the specifications of the functions and data structures that define it. For example, the Divide-and-Conquer Archetype represents the solution strategy where a programmer, when faced with a problem, takes the problem and breaks it down into some number of independent subproblems that are smaller than, but of the same type as, the original problem. These smaller subproblems are then recursively broken down further until a we have a subproblem small enough to solve easily. Once we have a subproblem small enough to solve, we do so. We then take the answers to all of the small subproblems, and merge their answers until we have the answer to the large problem that we originally wanted to solve. Examples of the Divide-and-Conquer Archetype can be found in such algorithms as Mergesort, Quicksort, and the Skyline problem [3].

Why are archetypes useful?

Archetypes provide a good way to teach programming because they naturally emphasize software engineering principles in a format that is language-independent and problem-independent. The archetypes not only discuss questions of algorithm development; they also address efficiency, proof methods, and test suite design. Even though the Divide-and-Conquer strategy is not new [3], the archetype format presents it in a way that has more learning utility than other approaches.

Archetypes are also useful because they present a systematic approach to solving a problem in computer science. Instead of presenting the student with just a problem and its solution, archetypes allow us to present the student with the entire method that leads from problem to solution.

Using a library of archetypes we can reduce the development of parallel and sequential algorithms to two steps [4]. The first step is choosing an archetype that manifests traits similar to the problem we want to solve. The second is to instantiate the chosen archetype to get a specific algorithm. Using an archetype we can prove the correctness, evaluate the performance or develop a test-suite for a particular application just as we developed the algorithm for the application. So, even though using an archetype to solve a problem does require some creative steps on the part of the programmer, the programmer now has the advantage of a structured method for writing a good algorithm.

The eText Interactive Environment

Considering that future readers of our book are familiar with the format of a usual textbook, we've chosen this format as a basis for eText. However, we expanded this format and made the structure of our electronic textbook (shown in Figure 1) more tree-like. eText has a three-level hierarchy of documents. Different levels correspond to the different types of presented information - many archetypes, many applications for a particular archetype and many program codes for a particular application. These levels are analogous to the chapters, sections, and subsections of a standard textbook respectively. Throughout the remainder of this paper, we will use the term "module" to refer to the documents containing archetypes, applications, and programs.

The highest level of generality contains the archetypes themselves. eText will incorporate chapters about Divide-and-Conquer, Dynamic Programming, Greedy Algorithms, and Hill-Climbing Methods. The next level contains the application modules, which resemble the archetype modules in both appearance and structure. While the archetype modules focus on teaching the archetype, the application modules present a specific problem, and then describe step-by-step how an algorithm to solve that problem can be created using the archetype. The final level is the actual source code. A code library is being built that contains code for various problems in a number of programming languages (such as C, Pascal, Fortran, Maisie [5] and -Fortran M [6]). A code viewer allows the user to select the problem, the language, and the archetype, and then see the code that solves the selected problem using the selected archetype, written in the selected language. This allows the user to see how a large variety of problems can be solved using a specific archetype, and it will also demonstrate the fact that a problem can be successfully solved using more than one archetype, but not all archetypes will produce good solutions for a specific problem. A simple editing environment will be included, which will allow the user to create, edit, compile, and run programs in several different sequential and parallel languages, so that the user can see how changing the various components of an instantiated algorithm affects the execution of the algorithm.

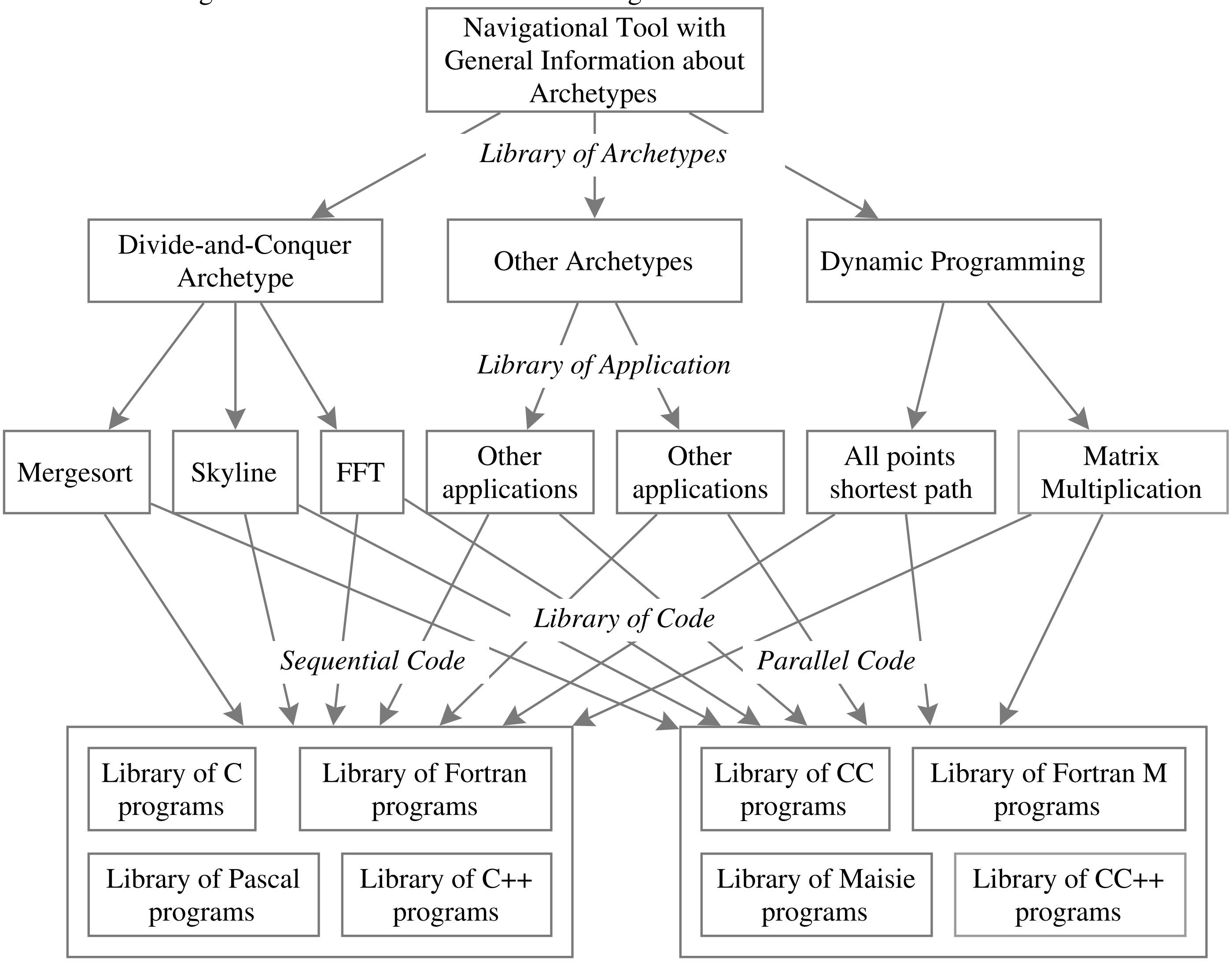

Figure 1. *The structure of eText.*

The central part of all eText modules is a multisectional text discussing the different facets of an archetype. Even though this text can be used as a stand-alone textbook, eText takes advantage of the computer's capabilities by employing multimedia in the form of sounds, animation, static graphics, and interactive figures to make its representation more useful and easier to understand.

Since the package was designed to be useful to audiences with varying levels of computer science background, we wanted the modules to have a format that allowed the user to determine how much explanatory material was necessary. A person who is unfamiliar with the idea of archetypes might want to go through and use the module as a full electronic textbook, while a person with more knowledge may not require the supplementary teaching material, and may wish to use the module for reference or review. With this in mind, the modules were designed to allow for two different levels of instructional depth, "Brief" and "Introductory." The Brief mode contains text written as a concise formal description of the archetype or problem with minimal detail. This amount of information is sufficient for experienced programmers and people already familiar with archetypes. The text of the "Introductory" mode includes informal descriptions, examples, interactive figures and animation, which are useful for beginners.

As we pointed out earlier, for the ease of navigation, the interface and the structure of the text is consistent throughout the chapters. Both tracks in any archetype consist of eight sections:

1) **Basic Idea**: A description of the solution strategy presented by the archetype.
2) **Algorithm Development**: A description of how one would develop an algorithm using the archetype, with a list of the functions and procedures of the archetype and their specifications.
3) **Algorithm**: A presentation of the general form of the algorithm.
4) **Parallel Implementation**: A generalization of the archetype with regard to parallel computing, which discusses how an algorithm is implemented on a parallel machine, and any specific points or subtleties that arise when using a parallel computer.
5) **Reasoning**: A discussion of how one would prove a developed algorithm.
6) **Testing**: A discussion of how to best test an algorithm designed using the archetype.
7) **Performance Analysis**: A discussion of the efficiency and complexity of the algorithms based on the archetype's strategy, with special emphasis placed on how the efficiency and complexity vary between the sequential and parallel case.
8) **Applications**: A list of various instantiated algorithms that use the archetype's method.

The electronic textbook shown in Figure 2 is the main window for an archetype. All of the chapters were designed to share the same general layout so that the interface for all of the chapters would become familiar to the user. The main text, which comprises the lesson for each archetype, is located in a text window along the left half of the window. The main window includes also several objects for navigational purposes. The contents of the main window are:

1) The main text window that contains the written presentation of the module. In addition to the two reading modes, the user is given further control of the depth of the material through the use of hyperlink [7] buttons that are placed throughout the text of the Introductory track, controlling the secondary windows that contain supplementary diagrams and interactive figures (examples of secondary windows opened by hyperlink buttons are shown in Figures 3, 4, and 5).

The supplementary figures are a useful learning tool, but could be tedious to someone who knows the material or who only wants a brief description of the ideas for reference while programming. With this in mind, the modules are structured so that all supplementary figures are user-activated and completely optional. This system allows the user to view the material with as little narration as desired, and insures that the user will not have to view any supplementary material that emphasizes ideas that he already understands.

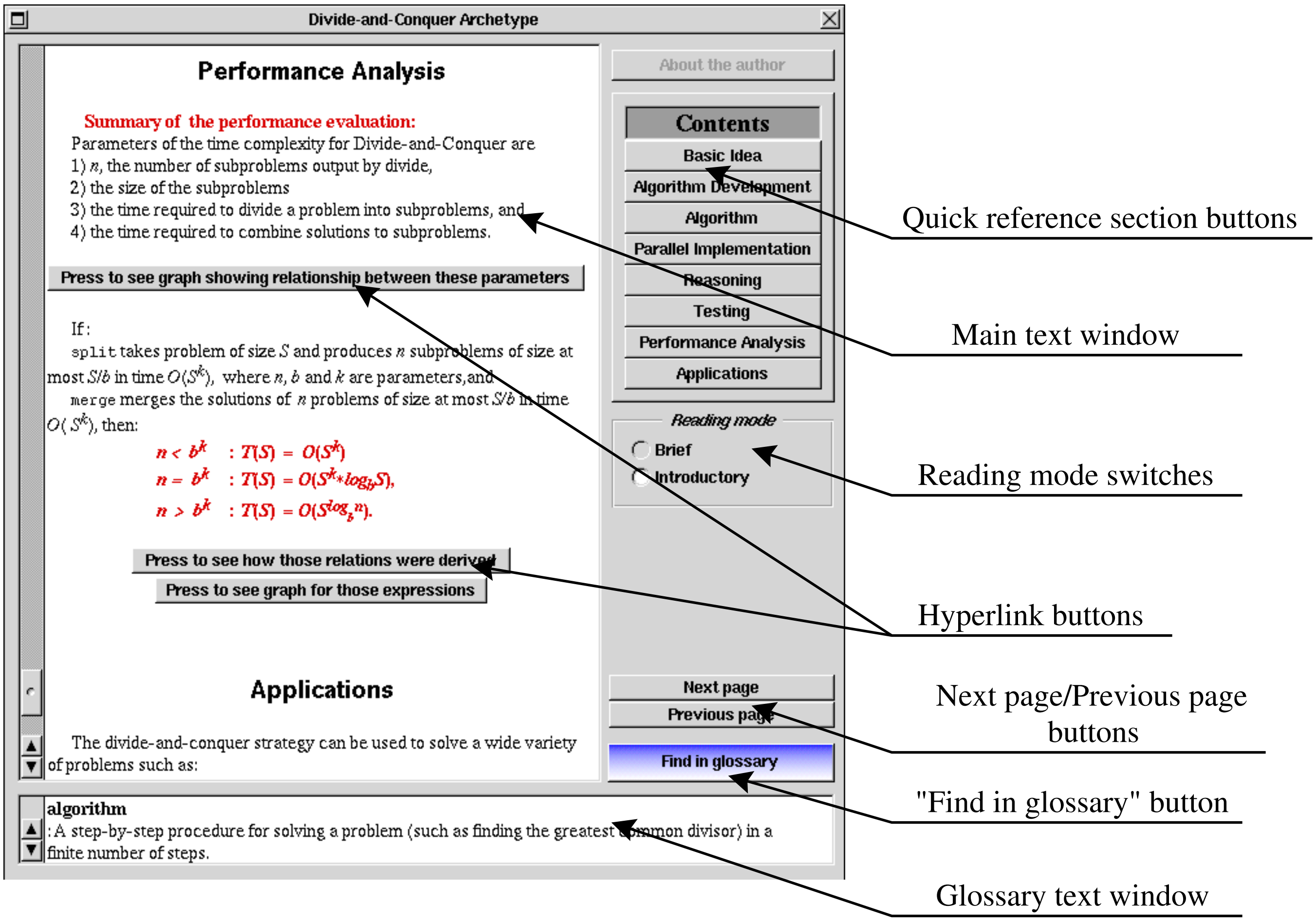

Figure 2. *The features of the electronic textbook.*

2) Quick reference section buttons, which form the table of contents for the module. Each button is labeled with the name of a section. When the user presses a table of contents button, the text automatically scrolls to make the corresponding section visible.

3) Reading mode switches that allow the user to determine the depth of the text by choosing the "Brief" or "Introductory" mode.

4) Next page/Previous page buttons that provide quick paging of the main text window.

5) The "Find in glossary" button and glossary text window that are included in case a user is not familiar with the document's computer science terminology. Pressing the "Find in glossary" button causes the definition of the word selected in any of the chapter's text windows to appear

in the glossary text window.

Our first implementation of the format and hierarchy described above was the production of the Divide-and-Conquer Archetype, and several application modules for that archetype. We present the details of this module to illustrate the electronic textbook.

Case Study: the Divide-and-Conquer Archetype

The Divide-and-Conquer chapter includes modules containing applications for Mergesort and Skyline [3]. For each module the text of both the brief and introductory reading modes consist of the same collection of 8 sections that cover similar aspects of the Divide-and-Conquer Archetype and its applications. The titles of the sections are reflected in the titles of the quick reference buttons shown in Figure 2. The sections for the Divide-and-Conquer Archetype are:

1) **Basic Idea**. This section presents the main idea of solving small problems directly by using some known simple algorithm and splitting large problems recursively, until they are small enough to be solved directly. In the introductory mode this section contains an informal example of using the Divide-and-Conquer strategy for moving and sorting the books in the library, supported by two slide shows illustrating moving the library and the sorting of numbers using Mergesort. The purpose of this example is to demonstrate the concepts of the archetype in an informal way. In the brief mode, this section contains only a few sentences description of the main idea without any examples or supporting slide shows.

2) **Algorithm Development**. This section presents the specifications of the primary functions, which can be instantiated to develop an algorithm:

size(in P), which returns the size of the problem P;
base_case(in P), which returns the boolean value *true* if and only if the problem P is a base-case problem;
find_base_case_solution(in P; out S), which produces the correct solution S to the base-case problem P ;
split(in P; out P_1, P_2, ..., P_n, Other), which splits the problem P into the subproblems P_1, P_2, ..., P_n and some other information Other that will be used in the *merge* procedure;
merge(in S_1, S_2, ..., S_n, Other; out S), which correctly merges subproblems S_1, S_2, ..., S_n and some other information Other into the solution S to the original problem P;
constant *base_case_size,* which is the size of the largest base-case problem.

The brief and introductory modes for this section differ in that there are more informal explanations and two supporting slide shows in the introductory mode. Figure 3 presents the last slide of one of the slide shows. This slide show emphasizes the basic steps in algorithm development for a problem determined to fit the Divide-and-Conquer pattern, and demonstrates how the sequential algorithm works for the general problem. Part of this slide show focuses the reader's attention on the archetype's functions and their properties. As all slide shows in the modules, this slide show is accompanied by voice narration, which can be switched on or off using the "Sound" switch.

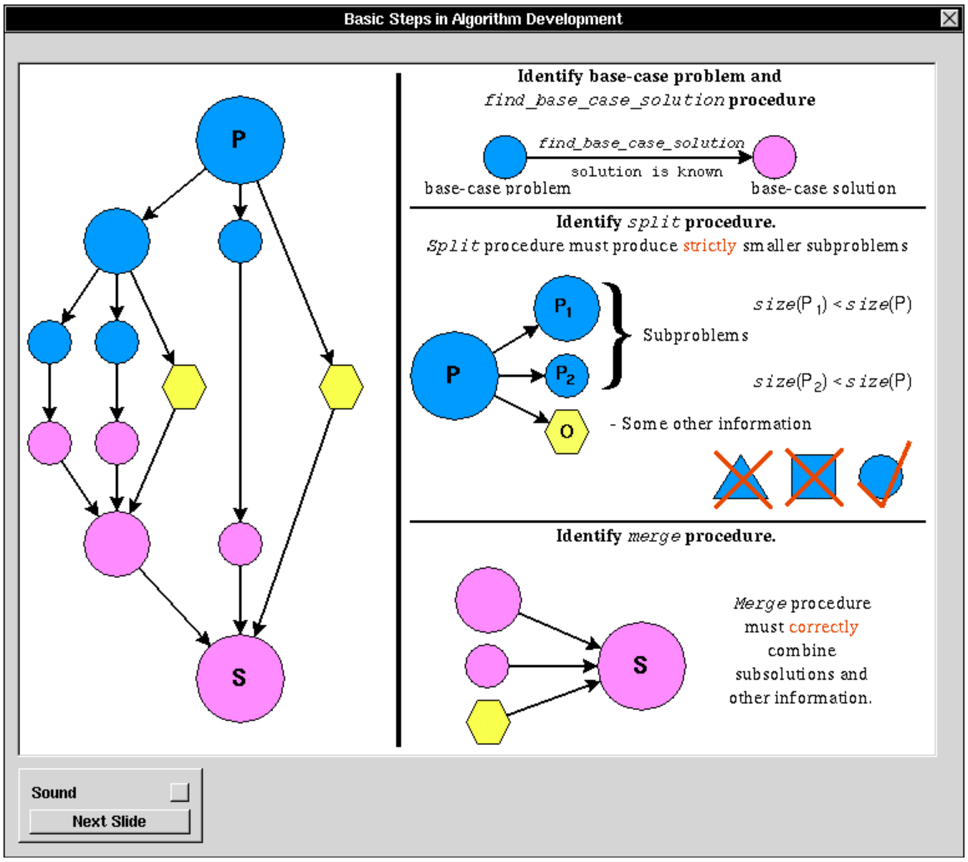

Figure 3. *Supporting slide show describing basic steps in the Divide-and-Conquer algorithm development.*

3) **Algorithm**. This section presents the language-independent description of the Divide-and-Conquer Archetype in terms of the functions and procedures developed in the preceding section.

```
procedure divide_and_conquer(in P;  out S)
begin
    if base_case(P) then base case
            find_base_case_solution(in P; out S)
    else begin larger than base case
            split(in P; out P1, P2, ..., Pn, Other);
            divide_and_conquer(in P1; out S1);
            .....
            divide_and_conquer(in Pn; out Sn);
            merge(in S1, S2, ..., Sn, Other; out S);
    end
end;
```

In addition to this description, the introductory text of this section includes a slide show that shows how the Divide-and-Conquer Archetype can be instantiated for the example of moving and sorting the books in a library.

4) **Parallel Implementation.** This section addresses the issues of algorithmic development, proof of correctness and performance evaluation, specifically for the parallel implementation of the Divide-and-Conquer Archetype. In particular, this section discusses the two main ways that the Divide-and-Conquer method can be implemented in parallel:

1) An implementation where the split, merge, and solving of subproblems are all done in parallel.

2) An implementation where the splitting and merging are done sequentially, while the solving of the subproblems is done in parallel.

5) **Reasoning.** This section discusses general methods of proving a Divide-and-Conquer algorithm using mathematical induction on the size of the problem. The proof of a Divide-and-Conquer algorithm requires the following steps:

- the proof of the correctness and termination of all functions and procedures in the algorithm.
- assuming that functions, *split*(), *merge*(), *base_case*() and *find_base_case_solution*() are correct, the correctness of the algorithm as a whole must be proved using mathematical induction. If problem *P* has size base_case_size or less it is a base-case problem. Therefore the function *base_case*() returns *true* and the function *find_base_case_solution*() returns the correct solution to the problem *P*.
- the induction hypothesis is stated, saying that the algorithm finds the correct solution to a problem of size k, where *k >= base_case_size*;
- then using the induction hypothesis and the fact that for any base-case problem, the algorithm returns the correct solution, the algorithm is proved to be correct for solving problem of size *k*+1.

6) **Testing**. This section contains general recommendations for the collection of test-suites for the developed algorithm. For Divide-and-Conquer, recommendations on the size of the test problems are made. All procedures and functions of the archetype must be tested separately and

together. Tests for the function *find_base_case_solution*() should include tests for all different categories of base-case problems. Tests for other procedures, *base_case*(), *split*() and *merge*(), should include problems of sizes,

- *base_case_size*,
- *base_case_size* + 1,
- *base_case_size* + 2,
- *n*base_case_size* and problems with larger size.

7) **Performance Evaluation**. This section addresses issues of performance evaluation, such as analyzing the performance of some general application of the Divide-and-Conquer Archetype based on the time-complexity of the *split* and *merge* procedures and size and number of subproblems produces by the *split* procedure. For most applications of the Divide-and-Conquer Archetype, the performance evaluation can be done just by using general formulas derived in this section defined by the values of the following parameters:

S is the size of the problem;
n is the number of subproblems the non base-case problem is split into;
b is such number that the size of the largest subproblem is S/b; and
k is such number that the time-complexity of the *split* and *merge* procedures is $O(S^k)$.

The formulas derived in this section defined in terms of the parameters listed above are:

$n < b^k : \mathrm{T}(S) = O(S^k)$;
$n = b^k : \mathrm{T}(S) = O(S^k * log_b S)$; and
$n > b^k : \mathrm{T}(S) = O(S^{log_b n})$.

Figure 4 represents the interactive figure that is part of the "Performance Evaluation" section. The purpose of this interactive figure is to demonstrate the difference between three time-complexity formulas derived in the section and allow readers to compare the same functions for different values of parameters. The interactive figure draws three time-complexity functions for each of the cases described in the archetype, and shows the user how these functions vary with respect to each other and with respect to three reference functions (the reference functions represent the common case where $n = b = 2$, and are always shown). The values of the parameters b and n can be entered using text-fields in the "Graph Inspector." Additionally, the inspector of the figure allows the user to choose which functions to draw and to choose option of removing or not removing old functions when drawing new ones.

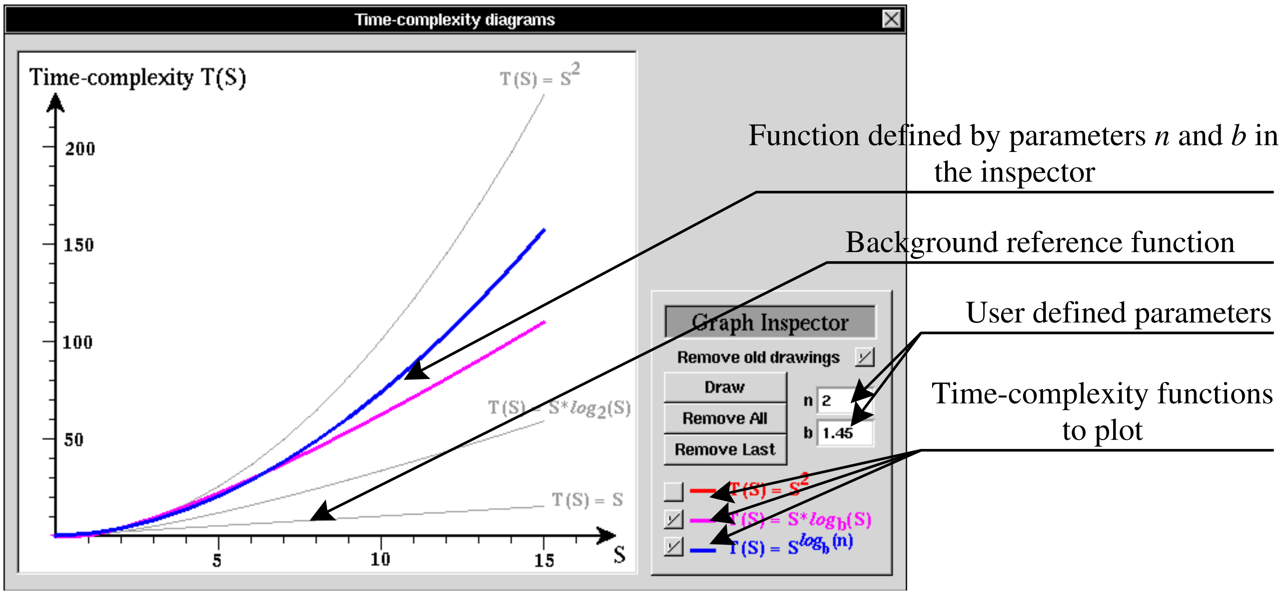

Figure 4. *Supporting interactive figure showing time-complexity diagram of the Divide-and-Conquer Archetype* .

8) **Applications**. This section lists the problems for which the Divide-and-Conquer Archetype can be used. There are such applications as Mergesort, Fast Fourier Transform, Nearest Neighbor and Skyline [3]. Chapters about Mergesort and Skyline are already developed.

Figure 5 shows the window containing the interactive figure from the chapter about the Skyline problem. The purpose of this figure is to explain the concept of the Skyline problem. This figure is part of the "Basic Idea" section of the chapter about the Skyline application of the Divide-and-Conquer Archetype. This figure allows the user to design a set of buildings using the mouse and edit the set using the "Building editor" part of the inspector. To illustrate the Skyline problem, the user's set of buildings is solved by the interactive figure so that the reader can see what outputs are created for given inputs. The process of solving the problem can be manipulated by buttons in the "Problem" part of the inspector. This interactive figure uses the actual Divide-and-Conquer algorithm to find the skyline for the entered set of buildings.

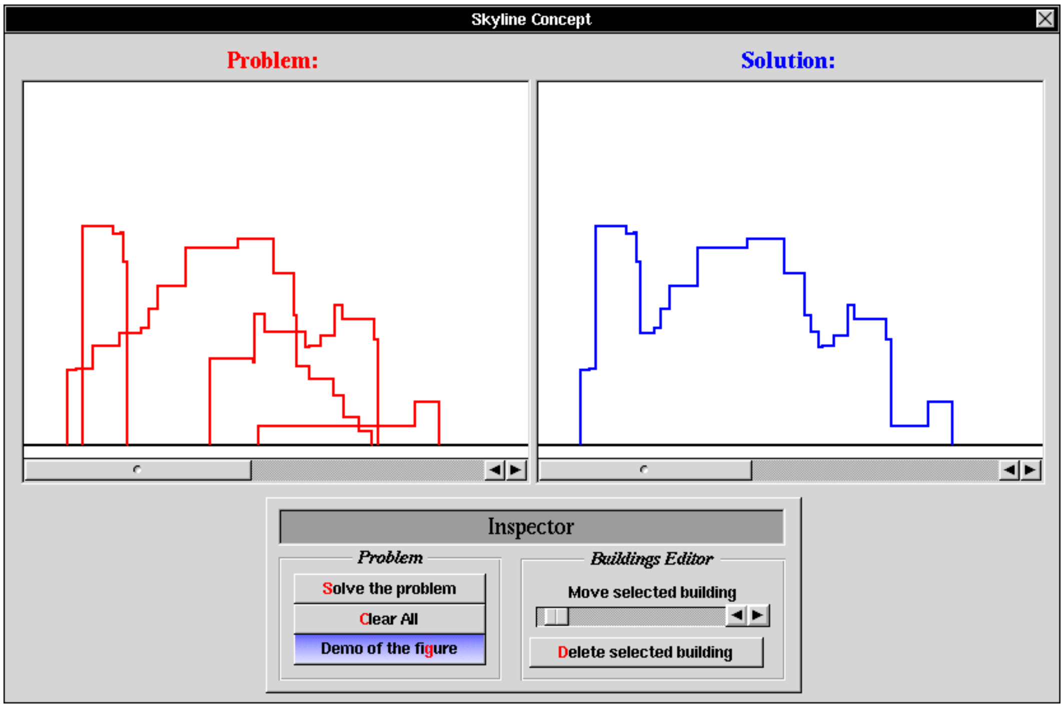

Figure 5. *Supporting interactive figure illustrating the inputs and outputs for the skyline problem.*

Even though the interactive figures and slide-shows in the eText modules let the student take a more active role in the learning process than a standard textbook, feedback on the material being taught is still very important to insure that the student fully understands the important ideas in each section. Therefore, we produced a quiz module, which presents the student with multiple choice questions that help reinforce the central ideas of each section, as well as subtle points that the student might not have recognized. For the chapter on the Divide-and-Conquer Archetype the feedback took the form of a review quiz using the same idea of the interactive environment (Figure 6.). The first questions review different aspects of the general archetype; and the rest of the questions focus on problem solving. The student is presented with the problem of string matching, and is then asked questions as to how he or she can solve the problem using the Divide-and-Conquer Archetype. Each incorrect answer results in an explanation of why the answer is incorrect, and each correct answer results in an explanation of why the answer is correct. Our experience indicates that these quizzes are proven to be a helpful supplement to the book; the questions force students reread some of the sections they didn't understand fully.

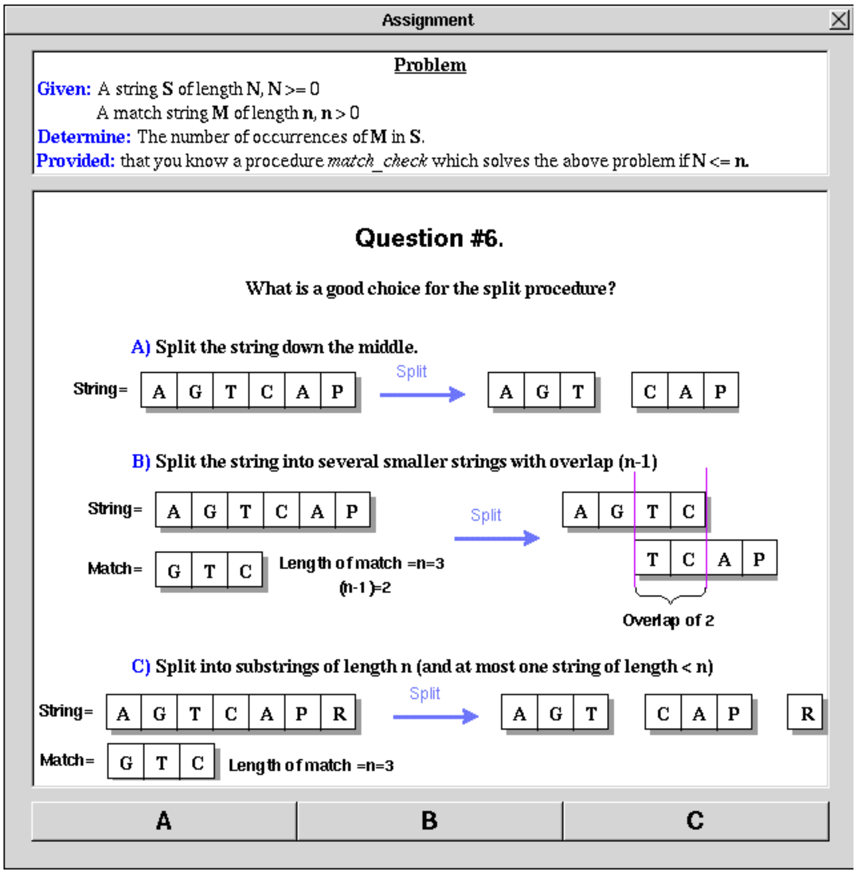

Figure 6. *Sample question from the Divide-and-Conquer quiz*

Conclusions

A unique feature of our electronic textbook is the content-driven material presented in the various modules. The idea of teaching using archetypes is a new one, but one that we feel has definite merit. Teaching archetypes not only emphasizes good software principles, but also provides the user with a relatively small arsenal of solution strategies that can help solve many computer science problems.

Even though the idea of using computers as learning tools is not new, nor is the idea of the electronic textbook, we believe that our approach to the electronic textbook to be innovative in its learning value. Instead of simply taking a textbook and transcribing the text from the book to an online word processor, we have created this electronic textbook from the ground up, taking full advantage of the power and versatility of the computer medium. We not only present the information in a way that no standard textbook could; we also provide the user with the opportunity to view code in several different parallel and sequential notation versions of C, C++, Pascal and Fortran. In the future we will provide a simple editing environment where the user can create, edit, compile, and run programs.

The modules discussed will be used this October to help teach the Computer Algorithms class at Caltech. We will use the students' input to further refine our modules. Since we strongly feel that using computers to teach computer science is a valid idea, we also plan to make modules that teach other archetypes (such as Branch-and-Bound and Greedy Algorithms), as well as other principles of computer science (such as Global Snapshots and Information Channels). Once we have several modules in a finished form, we will distribute them to universities. We intend for other schools to extend our work by developing modules of their own. All of our modules will remain non-copyrighted, so that they can be distributed and modified freely. We believe that the electronic textbook has incredible potential as an educational tool, and it is the ultimate goal of the eText project to see that this potential is realized.

Acknowledgements

The eText project is part of a larger effort, led by Mani Chandy at Caltech, to develop methods and tools to aid in the software engineering of parallel programs. The methods deal with the systematic development of parallel programs starting from specifications - and in many cases the specification is a sequential program that is required to be "parallelized." The tools support reasoning about parallel programs, and debugging on workstations, and then transporting parallel programs from workstations to parallel machines. Multimedia and archetypes play central roles in the overall effort. The development of eText technology was supported by ARPA under grant N00014-91-J-4014, and the development of eText chapters on scientific applications was supported by NSF under cooperative agreement CCR-9120008.

We would like to thank to Rohit Khare for his contributions to this paper and the project. We are also grateful to the following people who have been helpful consultants throughout the course of the eText project: Tal Lancaster, Berna Massingill, Paolo Sivilotti, and John Thornley.

Special thanks go to Adam Rifkin for his time and feedback, which contributed greatly to this paper.

This research is carried out under the supervision of Prof. K. Mani Chandy and we would like to thank hem for his guidance.

References

[1] R. Miller. *The Status of Parallel Processing Education: 1993*. State University of New York, Buffalo, 1993.

[2] K. M. Chandy and A. Rifkin. *An Archetype-Based Approach to Parallel Program Libraries*. Technical Report forthcoming, Computer Science Department, California Institute of Technology, 1993.

[3] B.M.E. Moret and H.D. Shapiro. *Algorithms from P to NP - Volume I: Design and Efficiency*, Benjamin/Cummings Publishing Company, 1991.

[4] A. Rifkin. *An Interactive Environment for Teaching Parallel Programming*. Technical Report TR-CS-93-13, Computer Science Department, California Institute of Technology,

1993.
[5] R. L. Bagrodia. *Designing Efficient Simulations Using Maisie*. Proceedings of 1991 Winter Simulation Conference, December 8-11, 1991, Phoenix, Arizona, p. 243-247.
[6] I. T. Foster, K. M. Chandy. *Fortran M: A Language for Modular Parallel*. Technical Report MCS-P327-0992, Mathematics and Computer Science Division, Argonne National Laboratory, 1992.
[7] B. Cotton, R. Oliver. *Understanding Hypermedia*. Phaidon Press Ltd, 1993.