DOWNLOAD PDF PROGRAMMING MODELS

Chapter 1 : What is Mathematical Programming? - Eudoxus Systems Ltd

A Programming model refers to the style of programming where execution is invoked by making what appear to be library calls. Examples include the POSIX Threads library and Hadoop's MapReduce. [1] In both cases, the execution model is different from that of the base language in which the code is written.

Chapman and Michael Wong March 2, The high performance computing HPC community is heading toward the era of exascale machines, expected to exhibit an unprecedented level of complexity and size. The community agrees that the biggest challenges to future application performance lie with efficient node-level execution that can use all the resources in the node. These nodes might be comprised of many identical compute cores in multiple coherency domains, or they may be heterogeneous, and contain specialized cores that perform a restricted set of operations with high efficiency. In general, heterogeneity and manycore processors are both expected to be common. Although we anticipate physically shared memory within each node, access speeds will vary considerably between cores and between types of memory, imposing deeper memory hierarchies and more challenging NUMA effects for performance optimizations. A critical challenge for using the massive parallel resources is the provision of programming models that facilitate the expression of the required levels of concurrency to exploit all of the hardware resources in the node, while permitting an efficient implementation by the system software stack. A programming model sits between the application and the hardware architecture. The design of language features and interfaces must agree on the details of the abstractions. In this article, we summarize a comprehensive list of features for parallel programming models to support recent and future heterogeneous and manycore architectures. We then make comparisons of several programming models against these features. A model should allow users to specify different kinds of parallelism that could easily be mapped to parallel architectures and that facilitate expression of parallel algorithms. At least four parallelism mechanisms should be considered for comparison: Recent accelerator-based architectures attach computational devices as coprocessors and rely on offloading model to exploit their capabilities. Optimizing parallel applications on shared memory ccNUMA machines is challenging. The effects of cache coherence, e. A programming model could help in this aspect by providing: A programming model should provide constructs for supporting coordination between various parallel work units, for example barrier, reduction and join operations for synchronizing parallel threads or tasks, point-to-point signal and wait operations to create pipeline or workflow executions of parallel tasks, and phase-based synchronization for streaming computations. Interfaces such as locks and mutexes are still widely used for protected data access. A model should provide language constructs for easily creating exclusive data access mechanism needed for parallel programming, and should define appropriate semantics for mutual exclusion to reduce the opportunities of introducing deadlocks. Architectural changes such as transactional memory provide alternatives to achieve similar data protection, which could also be part of the interface of a parallel model. Error handling provides support for dealing with faults from the user program or the system to improve system and application resilience. Support for tools, e. While functional languages can provide a cleaner abstraction for concurrency, it is not easy to rewrite all legacy code and library to a new base language. Ideally, a model would support at least these three languages. CUDA only for NVIDIA GPU and OpenCL are considered as low-level programming interfaces for recent manycore and accelerator architectures that can be used as user-level programming interfaces or intermediate-level interfaces for the compiler-transformation targets of high-level interfaces. The recent OpenACC standard created as a high-level interface for manycore accelerators helps users gain early experience of directive-based interfaces. Intel TBB and Cilkplus are task based parallel programming models used on multi-core and shared memory systems that have quality implementations and commercial support as well as open-source implementations. OpenMP is a comprehensive, well-developed standard that has been driven by industry, government labs and academia. It has multiple commercial and quality open-source implementations supporting hardware from many vendors and much existing scientific code is already using it. Comparisons The comparisons are shown in Figure 1 and 2. For parallelism support, asynchronous tasking or threading is still the foundational parallel mechanism that

is supported by all the models, and data parallelism such as OpenMP worksharing can be implemented using the basic asynchronous tasking and join synchronization. Overall, OpenMP provides the most comprehensive set of features to support a wide variety of parallelism patterns and architectures, on both host and devices, while others concentrate on support parallelism on either host or device only. Each of the programming models that support manycore architectures has its own way of organizing the massive threading capabilities x into a multiple-level thread hierarchy, e. Models that support devices and offloading computation provide constructs to specify data movement between discrete memory spaces, models that do not support other compute devices do not require them. Of the three commonly used synchronization operations, i. Note that since Cilk and Intel TBB emphasize tasks rather than threads, the concept of a thread barrier makes little sense in their model, so its omission is not a problem. Locks and mutexes are still the most widely used mechanism for providing mutual exclusion. In some cases vendor or third party profiling tools also have explicit support for OpenMP analyses. Comparison of heterogeneous and manycore programming models â€" Synchronizations, Mutual exclusions, Language binding, Error handing and Tool support Conclusion From the comparison, OpenMP clearly supports the most comprehensive set of features. It has evolved rapidly such that now it supports the emerging heterogeneous and manycore architectures including accelerators, as well as the conventional shared memory SMP, NUMA and multicore systems. The unique directive-based approach of OpenMP, which the OpenACC model for accelerators also borrows, enables a productive parallel programming model that significantly reduce migration and porting efforts for applications since it does not require that they be rewritten in a new language. OpenMP is the only directive based specification that allows the exploitation of the parallelism available in both the multiple CPUs in a host node and in attached processors using a single language. While a specification, whether a de facto standard or a formal standard, defines the interfaces for writing parallel programs, it is only used and adopted when there are quality implementations. To the best of our knowledge at the time of writing, the latest OpenACC standard, version 2. Pathscale is also working aggressively to release a compiler in the near future that supports the latest version of both OpenACC and OpenMP. However, a wide variety of users still use the proprietary CUDA model despite its productivity challenges because it currently delivers higher performance than the high-level programming models in the places where it is available. Thus the existence of multiple programming models, each having its own unique set of features that serve the specific needs of users and applications, and each having different degree of tradeoff between productivities and performance, is still necessary. About the Authors Dr. Starting from his Ph. Her research group has developed OpenUH, a state-of-the-art open source compiler that is used to explore language, compiler and runtime techniques, with a special focus on multi-threaded programming. He holds a B.

Chapter 2 : What is the Application Programming Model (APM)? - Definition from Techopedia

Programming models bridge the gap between the underlying hardware architecture and the supporting layers of software available to applications. Programming models are different from.

Conclusion Children from families in which English is not the language of the home represent a rapidly increasing percentage of students enrolled in U. Language minority students can be found in schools across the country, not just those in large cities or in areas near the U. All schools must be prepared to meet the challenge of an increasingly diverse student population, including many students who are not proficient in English. The effectiveness of various program models for language minority students remains the subject of controversy. Although there may be reasons to claim the superiority of one program model over another in certain situations Collier ; Ramirez, Yuen, and Ramey , a variety of programs can be effective. The choice should be made at the local level after careful consideration of the needs of the students involved and the resources available. ESL program models ESL programs rather than bilingual programs are likely to be used in districts where the language minority population is very diverse and represents many different languages. ESL programs can accommodate students from different language backgrounds in the same class, and teachers do not need to be proficient in the home language s of their students. ESL pull-out This is generally used in elementary school settings. Students spend part of the school day in a mainstream classroom, but are pulled out for a portion of each day to receive instruction in English as a second language. ESL class period This is generally used in middle school settings. Students receive ESL instruction during a regular class period and usually receive course credit. They may be grouped for instruction according to their level of English proficiency. The ESL resource center This is a variation of the pull-out design, bringing students together from several classrooms or schools. The resource center concentrates ESL materials and staff in one location and is usually staffed by at least one full-time ESL teacher. These programs are most easily implemented in districts with a large number of students from the same language background. Early-exit bilingual programs These are designed to help children acquire the English skills required to succeed in an English-only mainstream classroom. Instruction in the first language is phased out rapidly, with most students mainstreamed by the end of first or second grade. Two-way bilingual programs Also called developmental bilingual programs, these group language minority students from a single language background in the same classroom with language majority English-speaking students. Instruction is provided in both English and the minority language. Students serve as native-speaker role models for their peers. Two-way bilingual classes may be taught by a single teacher who is proficient in both languages or by two teachers, one of whom is bilingual. Other program models Some programs provide neither instruction in the native language nor direct instruction in ESL. However, instruction is adapted to meet the needs of students who are not proficient in English. Sheltered English or content-based programs These group language minority students from different language backgrounds together in classes where teachers use English as the medium for providing content area instruction, adapting their language to the proficiency level of the students. They may also use gestures and visual aids to help students understand. Although the acquisition of English is one of the goals of sheltered English and content-based programs, instruction focuses on content rather than language. As in sheltered English and content-based programs, English is taught through the content areas. Most students are mainstreamed after 2 or 3 years. Conclusion Successful program models for promoting the academic achievement of language minority students are those that enable these students to develop academic skills while learning English. The best program organization is one that is tailored to meet the linguistic, academic, and affective needs of students; provides language minority students with the instruction necessary to allow them to progress through school at a rate commensurate with their native-English-speaking peers; and makes the best use of district and community resources. Digest based on an article in August, Streamlined Seminar vol. National Association of Elementary School Principals. A Synthesis of studies examining long-term language minority student data on academic achievement. Bilingual Research Journal, 16, p. Education of linguistically and culturally diverse students: Educational practice report number 1. ED Lucas T. Promoting

the success of Latino language minority students: An Exploratory study of six high schools. Harvard Educational Review, 60 1 , Different types of ESL programs. Effective instruction for language minority students: An early childhood case study. Early Childhood Research Quarterly, 6, Longitudinal study of structured English immersion strategy, early-exit, and late-exit transitional bilingual education programs for language-minority children. A Descriptive study of significant features of exemplary special alternative instructional programs.

Chapter 3 : JUnit 5 User Guide

A rural public library in Canada's Yukon territory partnered with a local pool to get kids reading and swimming over summer break.

The picture below gives an overview of the endend stack: Such data model can be enriched with domain-specific annotations â€" e. Analytics, search, SAP Gateway and more. Different development guides are available on the SAP Help Portal to help you getting started with the development of the different application types. But note that the programming model is still evolving and further major deliveries are on their way. Read more on this in the Outlook section of this blog. Recommendations You may have already seen the slide below somewhere â€" e. If not yet, then check it out here! The slide above provides a list of recommendations meant to guide you for a best preparation for the future. Let me restructure the different recommendations a bit and briefly elaborate on them. Such an authorization access is declared once for a given CDS view and automatically used every time the view is consumed â€" everywhere. It offers a separation of matters by separating UI metadata from the back-end relevant metadata. BOPF concepts like actions, determinations, validations and authorization checks are still valid. For these service implementation options, the SAP Gateway runtime offers generically the read-only OData calls including goodies like sorting, filtering and text search for CDS views out of the box. This option is also to be used for cross BO scenarios. Manual implementation of read-only OData calls to Database Instead of implementing yourself the read-only OData calls again and again when building SAP Gateway service in transaction SEGW, use the mapped data source or -better- the referenced data source option instead. Business logic mixed with technical aspects e. For example, BOPF offers dedicated exits for authority checks. Business logic mixed with protocol specific APIs e. Easy reuse of modularized and decoupled custom business logic for implementing business object behaviours in different applications and scenarios. Lower TCD for the future: The main investment areas of the future development are: Find this information here.

Chapter 4 : calendrierdelascience.com Different Programming Models

Example Programming Models St. Francis Xavier University - Describes their use of the wellness wheel and their programming points system. 4 Point Programming model - Description of points system, student concerns calendar, programming ideas and more.

What is Mathematical Programming? Introduction First a definition: Mathematical Programming MP is the use of mathematical models, particularly optimizing models, to assist in taking decisions. It is still used, for instance, in oil refineries, where the refinery programmers prepare detailed schedules of how the various process units will be operated and the products blended. Mathematical Programming is, therefore, the use of mathematics to assist in these activities. Mathematical Programming is one of a number of OR techniques. Its particular characteristic is that the best solution to a model is found automatically by optimization software. Being so ambitious does have its disadvantages. Mathematical Programming is more restrictive in what it can represent than other techniques. Nor should it be imagined that it really does find the best solution to the real-world problem. It finds the best solution to the problem as modelled. If the model has been built well, this solution should translate back into the real world as a good solution to the real-world problem. If it does not, analysis of why it is no good leads to greater understanding of the real-world problem. The key characteristics of MP are shown in the diagram below. What can Mathematical Programming Do? The essential characteristics of a problem for Mathematical Programming to be applied are: These are reflected in the essential components of an MP model: The definitions of all these components will change repeatedly during the building of the model. Although the process of MP involves finding optimum solutions, nobody is suggesting that the solution is optimum to the real-world problem. If the model is reasonably faithful, then its optimum solution should be a good solution to the real-world problem. Whether it is or not, the process of building the model and analysing the solutions is a very powerful tool in analysing the real-world problem. Mathematical Programming is very suitable for problems involving blending, continuous flow processing, production and distribution, and strategic planning. It answers questions such as: In an LP model all the relationships are linear, hence the name. LP has been so successful for two reasons: Problems involving planning, blending, production and distribution are all capable of being solved using Linear Programming. The principles of MP also apply to problems involving logistics and scheduling but the processes of tackling such problems are more varied and a mixture of techniques is likely to be used, including MP, heuristics and special-purpose algorithms. Building an MP Model Much of the art of building an MP model revolves around deciding which aspects of a real-world problem should be included and which should not. In practice this is an iterative process. To start with, keep things simple. If in doubt, leave out. If the optimum solution from the resulting model is clearly wrong in the real world, add extra detail and try again. When starting to formulate a model, it may be helpful to think in terms of the typical decision variables of an MP model.

Chapter 5 : Program Models | Programming Librarian

Parallel programming models are closely related to models of computation. A model of parallel computation is an abstraction used to analyze the cost of computational processes, but it does not necessarily need to be practical, in that it can be implemented efficiently in hardware and/or software.

Services can also be any compiled executable program written in any language or code running in a container hosted on a Service Fabric cluster. Guest executables A guest executable is an existing, arbitrary executable written in any language that can be run as a service in your application. However they still benefit from features the platform offers, such as service discoverability, custom health and load reporting by calling REST APIs exposed by Service Fabric. They also have full application lifecycle support. Get started with guest executables by deploying your first guest executable application. Containers By default, Service Fabric deploys and activates services as processes. Service Fabric can also deploy services in containers. Service Fabric supports deployment of Linux containers and Windows containers on Windows Server Container images can be pulled from any container repository and deployed to the machine. You can deploy existing applications as guest executables, Service Fabric stateless or stateful Reliable services or Reliable Actors in containers, and you can mix services in processes and services in containers in the same application. Learn more about containerizing your services in Windows or Linux Reliable Services Reliable Services is a light-weight framework for writing services that integrate with the Service Fabric platform and benefit from the full set of platform features. Reliable Services provide a minimal set of APIs that allow the Service Fabric runtime to manage the lifecycle of your services and that allow your services to interact with the runtime. The application framework is minimal, giving you full control over design and implementation choices, and can be used to host any other application framework, such as ASP. Reliable Services can be stateless, similar to most service platforms, such as web servers, in which each instance of the service is created equal and state is persisted in an external solution, such as Azure DB or Azure Table Storage. Reliable Services can also be stateful, exclusive to Service Fabric, where state is persisted directly in the service itself using Reliable Collections. State is made highly-available through replication and distributed through partitioning, all managed automatically by Service Fabric. Learn more about Reliable Services or get started by writing your first Reliable Service. NET Core is a new open-source and cross-platform framework for building modern cloud-based Internet-connected applications, such as web apps, IoT apps, and mobile backends. Service Fabric integrates with ASP. Learn more about ASP. Reliable Actors Built on top of Reliable Services, the Reliable Actor framework is an application framework that implements the Virtual Actor pattern, based on the actor design pattern. The Reliable Actor framework uses independent units of compute and state with single-threaded execution called actors. The Reliable Actor framework provides built-in communication for actors and pre-set state persistence and scale-out configurations. Because Reliable Actors is an application framework built on Reliable Services, it is fully integrated with the Service Fabric platform and benefits from the full set of features offered by the platform.

Chapter 6 : Resident Assistant â€“ Programming Models

Dave Stearns Microsoft Corporation. June Introduction. Every component developer has to design a programming model. When you write a Component Object Model (COM) control or dynamic-link library (DLL), you must decide how that component will be programmable or, in other words, how developers will write code to manipulate that component.

Presets are defined by selecting items from Lighting, Keypads, Interfaces, Global, etc. The properties tab allows you to add more features to a button cycle dim, double-tap, etc. LED Logic may be set to any of the following types: Room - LED is on true when at least one of the zones programmed on the Press preset is on. Otherwise, the LED is off false. Room logic is typically used as a monitoring function i. Are any of the basement lights on? Scene - LED is on true when all of the zones are at the exact levels programmed on the Press preset. Otherwise the LED is off false. Scene logic is typically used to tell what state a room is in i. Is the Family Room in the Entertain Scene? Pathway - LED is on true when all of the zones programmed under the Press preset are on at any level. Pathway logic is typically used to tell whether or not a path is lit up. Is the Kitchen to MBR pathway lit up? Always On - LED is always on, regardless of what the button does. As soon as the button is released, the LED is off. The default state of the LED is off. If power to the processor is lost and then restored, the LED will return to its previous state. Cycle Dim mode can be added to a button by checking the cycle dim box on the properties tab in the control station programming screen. Cycle Dim mode is only available for Default and Advanced Toggle programming models. By default, cycle dim mode operates as follows: When the keypad button is pressed for more than 0. When the button is released, the zones will stop raising and remain at the current intensity. When the button is pressed again, it will lower the intensity of the zones. When the button is released again, it will stop lowering. This alternation between raise and lower will continue until the level that the user desires is achieved. Four seconds after the last raise or lower operation, cycle dim mode is deactivated and the button returns to its normal operation.

Chapter 7 : History of Asynchronous Programming Models in .NET framework | Coding Canvas

Basic programming model. A library of static methods is a set of static methods that are defined in a Java class. A basic model for Java programming is to develop a program that addresses a specific computational task by creating a library of static methods, one of which is named main().

There are 3 models provided by asp. Each model has different scope and application of use, so I am going to explain about each model. Web Form This is basically an event driven programming model which lets you build dynamic websites using familiar drag- and- drop. Design view and lot many controls helps you to develop application rapidly. This is a tradition approach to build website in asp. Web Forms are compiled and executed on the server, which generates the HTML that displays the web pages. This Model basically targets to those developers who prefer declarative and control based programming, such as Win Form etc. Developer does not require a lot of experience to develop application in this model. It is the most popular model of ASP. NET but has been criticized for the lack of control over the generated markup because a lot of abstractions are present in this model. In this model each page having aspx and. So by maintaining two file this model separate code from front end tags for generating UI. There are following functionalities provided by this model. Use page controller pattern means each page has a code behind class that act as a controller. Code behind file depends on View so both will combine during execution. There is lots of server controls provided, so it is very easy and fast development possible with this model. With the basic knowledge of this you can start development in this model because all these things are abstract. This Model provides Rapid development 2. NET MVC is for developers who are interested in development like test-driven development, separation of concerns, inversion of control IoC etc. This framework separates the business logic layer of a web application from its presentation layer. The Model represents the behavior and data of application logic The View displays the data and information The Controller handles the input from user and create link between Model and View. Controller read data from a view and sends input data to the model. MVC helps you manage complex applications, because you can focus on one aspect of an application at a time. For example, you can focus on the view without depending on the business logic. It also makes it easier to test an application. The MVC also facilitates the group development. Different developers can work on the view, the controller logic, and the business logic in parallel or if I want to explain in easy word then I can say developers can work on Model, View and controller separately and later on they can merge whole application. NET Web Forms model. It is a lightweight, highly testable framework, integrated with all existing ASP. This model uses Front Controller pattern. There is a single central controller for all pages to process web application. There are following basic features of this model.

Chapter 8 : Parallel programming model - Wikipedia

Programming Model. You write code for your Lambda function in one of the languages AWS Lambda supports. Regardless of the language you choose, there is a common pattern to writing code for a Lambda function that includes the following core concepts.

Ignoring the non-standard types for a moment, all three bit pointer models satisfy the rules as specified. A change in the width of one or more of the C datatypes affects programs in obvious and not so obvious ways. There are two basic sets of issues: Programs depending on those relationships often cease to work properly. LLP64 preserves the relationship between int and long by leaving both as bit datatypes. Objects not containing pointers will be the same size as on a bit system. LLP64 is really a bit model with bit addresses. Most of the runtime problems associated with the assumptions between the sizes of the datatypes are related to the assumption that a pointer will fit in an int. To solve this class of problems int or long variables are changed to long long, a non-standard datatype. This solution is optimized for the first class of problem and is dependent on the introduction of a new datatype. Most of these interfaces have been stable for almost 25 years across all versions of the UNIX operating system. Thus, LLP64 requires extensive modifications to existing specifications to support those places which should naturally become bit wide. ILP64 attempts to maintain the relationship between int, long and pointer by making all three the same size as is the case in ILP Assignment of pointers to int or long variables does not result in data loss. This is a potential conflict with existing typedefs, and is especially contrary to the spirit of C development, which has avoided embedding size descriptions into fundamental datatypes. Programs that must preserve the size and alignment of data are forced to use the non-standard datatype and may not be portable. The world is currently dominated by bit computers, a situation that is likely to exist for the foreseeable future. These computers run 16 or bit programs, or some mixture of the two. Meanwhile, bit CPUs will run bit code, bit code, or mixtures of the two and perhaps even some bit code. Key issues facing the industry are the interchange of data between 64 and bit systems in some cases on the same system and the cost of maintaining software in both environments. Such interchange is especially needed for large application suites like database systems , where one may want to distribute most of the programs as bit binaries that run across a large installed base, but be able to choose bits for a few crucial programs, like server processes. ILP64 implies frequent source code changes and requires the use of non-standard datatypes to enable interoperability and maintain binary compatibility for existing data. LP64 takes the middle road. A bit type long is provided to support the full arithmetic capabilities and is available to use in conjunction with pointer arithmetic. Programs that assign addresses to scalar objects need to specify the object as long instead of int. Programs that have been made bit safe can be recompiled and run on bit systems without change. The datatypes are natural, each scalar type is larger than the preceding type. As a language design issue, the purpose of having long in the language anticipates cases where there is an integral type longer than int. The fact that int and long represent different width datatypes is a natural and common sense approach and is the standard in the PC world where int is 16 and long is bits. We argue that the Open Systems community, most particularly the application developers, are best served if there is a single choice widespread in the emerging bit systems. This removes a source of subtle errors in porting to a bit environment, and encourages more rapid exploitation of the technology options. Also, this is an opportune moment to make such an agreement, since the early shippers Digital and SGI have already selected the same model namely, LP64 , while other vendors have not yet committed their shipping products to a choice. The remainder of this paper describes the evaluation criteria we suggest using to make a selection for the industry, and assesses the LP64 and ILP64 models against these criteria. The LLP64 model is not, in our view, a satisfactory basis for widespread adoption and use since it requires extensive modification to existing standards. The investment in code, experience, and data surrounding these applications is the largest determiner of the rate at which new technology is adopted and spread. In addition, it must be easy for an application developer to build code which can be used in both existing and new environments. At this point, there is experience at Digital and SGI with the realities of porting applications to an LP64 based bit programming environment. The Digital and SGI

experiences prove complementary facts. Digitals shows that it is possible for ISVs and customers to port large quantities of code to a bit environment, while producing and inter-operating with bit ports elsewhere. SGIs experience shows that code can be improved to be compilable for either 32 or bits and still be able to interchange data in the more tightly-coupled fashion expected by processes on the same system. Although we are beginning to see a number of applications grow to the point of requiring the larger virtual address space there hasnt been a requirement for a bit int data type. The majority of todays bit applications previously ran only on bit systems usually some flavor of UNIX , and had no expectation of a greater range for the int data type. In such cases, the extra 32 bits of data in a bit integer are wasted. Future application requiring a larger scalar data type can use long. Other language implementations will continue to support a bit integer type. Such applications have been amongst the quickest to find reason to move to bit environments. Nearly all applications moving from a bit system require some minor modifications to handle bit pointers, especially where assumptions about the relative size of int and pointer data types were made. We have also noticed assumptions about the relative sizes of int, char, short and float datatypes that do not cause problems in an LP64 model since the sizes of those datatypes are identical to those on a bit system but do in a ILP64 model. A crucial investment for end-users is the existing data built up over decades in their computer systems. Any solution must make it easy to utilize such data on a continuing basis. This is likely to cause practical problems in producing code which can run on both 32 and 64 bit platforms without ifdef constructions. It has been possible to port large quantities of code to LP64 models without the need to make such changes, while maintaining the investment made in data sets, even in cases where the typing information was not made externally visible by the application. Most ints in existing programs can remain as 32 bits in a bit environment; only a small number are expected to be the same size as pointers or longs. Under LP64 very few instances need to be changed. These differences will predictably cause subtle hard-to-find bugs. These documents have developed over many years to codify existing practice and define agreement on new capabilities. Thus, the specifications collectively are a major value to the system developers, application developers and end-users. There is a body of work on verifying that implementations correctly embody the details of the specification and certify that fact to various consumers. These verification suites are also part of the "glue" that keeps us a community. Any bit programming model cannot invalidate large quantities of these specifications with their extensive detailed descriptions and expect to achieve wide adoption. Currently shipping, LP64 based operating system have met and passed many of the existing specifications and verification suites. It is our observation that there was no major philosophical barrier in doing so, but rather much detailed review of critical items buried within the specifications and verification suites. As a community, we know by demonstration that LP64 systems can comply with the commercially important standards; there is NO such demonstration today for LLP64 or ILP64 systems In particular, most standards, but particularly the language standards such as the ANSI C specification deliberately are silent in enforcing width decisions for basic data types since history has shown an assortment of choices made to reflect underlying architectures. This leads to deliberate ambiguity in the meaning of certain code samples, both as they move between different C compilers and when they move between optimization levels on specific C compilers. Some of these can occasionally cause practical problems, but well-written code and tools such as lint have eased this problem significantly. Moreover, the experience of porting applications between various vendor platforms has significantly helped find problems like this. Nonetheless, such examples exist both as demonstration points and as practical problems - they require significant intellectual energy to get them right. With LP64, we have the accumulated experience of several implementers and application developers to help. However, for some time the acceptable economic boundaries for many problems will be a barrier. Thus, crucial to rapid adoption is attention to inherent performance differences between the models. We see two categories of differences: Instruction cycle penalties will be incurred whenever additional cycles are required to properly implement the semantics of the intended programming model. However, compiler writers have extensive experience with the type of optimizations that are available. For example, in LP64 its only necessary to perform sign extension on int when you have a mixed expression including long; most integral expressions do not include longs. Compilers can be smart enough to only sign extend when necessary. Many current architectures Alpha, MIPS, Sparc V9 and PowerPC

dont have a problem because there is a bit load that performs sign extension, although they may have the analogous problem with bit unsigned int. Given that most CPUs will spend much of their time executing bit operations whether running bit programs, or simply doing bit operations in bit code , it seems difficult to understand why many implementations would penalize bit operations. In our porting experience, these "inner loop" issues have NEVER become major performance determiners for commercial applications, and the current balancing act between memory cycle times and CPU cycle times leaves open issue slots which can often absorb the additional cycles. A much larger practical effect in some commercially important applications comes from the consumption of additional memory and the costs of transporting that memory throughout the system. Further, the latency penalty can be enormous, especially to disk, where it can exceed 1,, CPU cycles 3 nsec to 3 msec. Some software vendors have experimented with an ILP64 model, which can be approximated on LP64 systems by changing all int declarations to long. In these cases, the conclusion reached after these experiments was not to use ILP64, since the application did not benefit from the additional range of int values and did not wish to pay the performance penalty of extra memory use. The vendors of these platforms have worked with most of the application developers whose support is critical to any widespread adoption of bits in the community. The practical problems of such ports have been resolved in a fashion that is demonstrably successful based on years of market experience. These attempts have not had the broad market base that LP64 systems have had, although they do demonstrate that it is feasible to complete the implementation of an ILP64 environment. Applications already modified for LP64 will need additional effort in code audit, changes to usage of int, and performance tuning to run on ILP64 systems. Many large ISVs would need separate code pools to support this difference. It is hard to see why this change provides them any value to compensate for the additional effort. Applications not yet modified for any bit systems have the experience of others to guide them - experience expressed as tools to identify troublesome constructs, and as porting documents. These experiences become the practical guidance needed to both encourage adoption and avoid pitfalls. SUMMARY Each of the evaluation criteria are subject to extensive further exploration a quote comes from Jim Gray - Computing is fractal; wherever you look, there is infinite complexity. We have argued that each issue supports a choice of LP Interoperability is improved by the ability to use a standard data type to declare data structures that can be used in both 32 and bit environments. Standards conformance has been demonstrated both in the practical sense of porting many programs and in the formal sense of compliance with important industry standards. Performance has been a major and effective selling theme for both LP64 systems, and the memory size penalty for unneeded bit integers can be very high for some applications. Transition from the current industry practice is smooth and direct following a path grooved with experience and demonstrated success. All this, as well as natural use of the native C datatypes to support all the widths needed in a bit system make a compelling argument for the inherent advantage of LP

Chapter 9 : Study Abroad - Program Models | USAC

The MVC programming model is a lighter alternative to calendrierdelascience.com Web Forms model. It is a lightweight, highly testable framework, integrated with all existing calendrierdelascience.com features, such as Master Pages, Authentication and Security.

Java has five additional primitive data types: A Java program is composed of statements, which define the computation by creating and manipulating variables, assigning data-type values to them, and controlling the flow of execution of such operations. Declarations create variables of a specified type and name them with identifiers. Java is a strongly typed language because the Java compiler checks for consistency. The scope of a variable is the part of the program where it is defined. Assignments associate a data-type value defined by an expression with a variable. Initializing declarations combine a declaration with an assignment to initialize a variable at the same time it is declared. Conditionals provide for a simple change in the flow of executionâ€”execute the statements in one of two blocks, depending on a specified condition. Loops provide for a more profound change in the flow of executionâ€”execute the statements in a block as long as a given condition is true. We refer to the statements in the block in a loop as the body of the loop. Java supports two additional statements for use within while loops: The break statement, which immediately exits the loop The continue statement, which immediately begins the next iteration of the loop For notation. Many loops follow this scheme: If a block of statements in a conditional or a loop has only a single statement, the curly braces may be omitted. The following table illustrates different kinds of Java statements. An array stores a sequence of values that are all of the same type. If we have N values, we can use the notation a[i] to refer to the ith value for any value of i from 0 to N Creating and initializing an array. Making an array in a Java program involves three distinct steps: Declare the array name and type. Initialize the array values. The default initial value is zero for numeric types and false for type boolean. We can specify the initialization values at compile time, by listing literal values between curly braces, separated by commas. Once we create an array, its size is fixed. A program can refer to the length of an array a[] with the code a. Java does automatic bounds checkingâ€”if you access an array with an illegal index your program will terminate with an ArrayIndexOutOfBoundsException. An array name refers to the whole arrayâ€”if we assign one array name to another, then both refer to the same array, as illustrated in the following code fragment. This situation is known as aliasing and can lead to subtle bugs. A two-dimensional array in Java is an array of one-dimensional arrays. A two-dimensional array may be ragged its arrays may all be of differing lengths , but we most often work with for appropriate parameters M and N M-by-N two-dimensional arrays. To refer to the entry in row i and column j of a two-dimensional array a[][], we use the notation a[i][j]. Static methods are called functions in many programming languages, since they can behave like mathematical functions. Each static method is a sequence of statements that are executed, one after the other, when the static method is called. Defining a static method. A method encapsulates a computation that is defined as a sequence of statements. A method takes arguments values of given data types and computes a return value of some data type or causes a side effect. Each static method is composed of a signature and a body. Invoking a static method. A call on a static method is its name followed by expressions that specify argument values in parentheses, separated by commas. When a method is called, its argument variables are initialized with the values of the corresponding expressions in the call. A return statement terminates a static method, returning control to the caller. If the static method is to compute a value, that value must be specified in a return statement. Java methods have the following features: Arguments are passed by value. When calling a function, the argument value is fully evaluated and the resulting value is copied into argument variable. This is known as pass by value. Array and other object references are also passed by value: Method names can be overloaded. Methods within a class can have the same name, provided they have different signatures. This features is known as overloading. A method has a single return value but may have multiple return statements. A Java method can provide only one return value. Control goes back to the calling program as soon as the first return statement is reached. A method can have side effects. A method may use the keyword void as its return type, to indicate that it has no return value and produces side effects consume

input, produce output, change entries in an array, or otherwise change the state of the system. A recursive method is a method that calls itself either directly or indirectly. There are three important rules of thumb in developing recursive programs: The recursion has a base case. Recursive calls must address subproblems that are smaller in some sense, so that recursive calls converge to the base case. Recursive calls should not address subproblems that overlap. A library of static methods is a set of static methods that are defined in a Java class. A basic model for Java programming is to develop a program that addresses a specific computational task by creating a library of static methods, one of which is named main. Libraries of static methods enable modular programming, where static methods in one library can call static methods defined in other libraries. This approach has many important advantages. Work with modules of reasonable size Share and reuse code without having to reimplement it Substitute improved implementations Develop appropriate abstract models for addressing programming problems Localize debugging Unit testing. A best practice in Java programming is to include a main in every library of static methods that tests the methods in the library. We use static methods from four different kinds of libraries, each requiring slightly differing procedures for code reuse. Standard system libraries in java. Imported system libraries such as java. An import statement at the beginning of the program is needed to use such libraries. Other libraries in this book. To use such a program, download the source from the booksite into your working directory or follow these instructions for adding algs4. The standard libraries that we have developed for use in this book. To use such a program, download the source from the booksite into your working directory or follow these instructions for adding stdlib. To invoke a method from another library, we prepend the library name to the method name for each call: