Application of Advanced Multi-Core Processor Technologies to Oceanographic Research

Mark R. Abbott
104 CEOAS Administration Building
College of Earth, Ocean, and Atmospheric Sciences Oregon State University
Corvallis, OR 97331-5503
phone: (541) 737-5195 fax: (541) 737-2064 email: mark@coas.oregonstate.edu

Grant Number N000141110104
Grant Number N000141210298
Grant Number N000141410199 (subaward from MBARI)
http://www.ceoas.oregonstate.edu

LONG-TERM GOALS

Improve our ability to sense and predict ocean processes, utilizing state-of-the-art information processing architectures and techniques.

OBJECTIVES

Next-generation processor architectures (many-core, multi-threaded) hold the promise of delivering enormous amounts of compute power in a small form factor and with low power requirements. However, new programming models are required to realize that potential. Our objectives are to deploy data processing and vehicle control systems onto a variety of "systems on a chip" (SoC) to provide autonomous functionality, to develop a platform for the dissemination, aggregation, and analysis of real-time sensor information, and to develop computational methods to detect and mitigate faults in autonomous vehicles.

APPROACH

We applied three themes in our approach, due to the multi-pronged nature of our recent work. With respect to the application of heterogeneous, multi-core SoCs, we continued along the arc of our previous work; we continue to enhance the compute device scheduling system, adding power-aware schedulers, and to consider system resilience and fault-tolerance. Our recent work on a real-time sensor platform, that we call SensorStream, required a different approach. For this work, our approach was to divide the system into three elements: the embedded sensor device, the cloud-based back-end, and the front-end web portal. Each of these components is independent, and they function through the use of a well-defined API. The fault detection and mitigation work requires yet another approach, and that is to work very closely with Ben Raanan, based at MBARI. As he is an expert in time series analysis, we are following his analyses in the development of automatic tools for writing classifiers that will detect failures and anomalous conditions.

WORK COMPLETED

In the few months since the beginning of the "Fault detection, diagnosis, and mitigation for long-duration AUV missions with minimal human intervention" grant, we traveled to the MBARI facility for a kickoff meeting in June. We set a course for the next year of the grant. We decided that, due to its current limitations, the best course of action would be to invest in the future of the project by taking the time to modernize and modularize the vehicle software. Brian Keift and M. Jordan Stanway are taking on the work of analyzing their code, and we are working on the Robot Operating System (ROS) and MOOS-DB systems to evaluate their pros-and-cons and determine whether they would be effective platforms for the new vehicle management system. We have succeeded in building these systems at our facility, and have shown that they can be used in a heterogeneous environment. Our next task in this area is to determine whether, and how, we can integrate our compute task dispatcher into these systems.

As part of the vehicle software modernization work, we are evaluating the on-vehicle simulator. The current simulator is tightly coupled with the vehicle logic, which limits the extent to which significant changes can be made. Furthermore, the fault and anomaly classification work requires that the same simulator is used on the vehicle and the classifier development tool. If these simulators differ, the classifiers may fail to detect true errors and may identify false errors. Rather than attempt to extract the current simulator from the existing vehicle code, Ben Ranaan found a simulator written in Matlab by Timothy Prestero at WHOI. We have been working on porting that simulator from Matlab into an Object-Oriented paradigm to integrate it into our tools, and eventually, vehicle.

Once the port of the simulator is complete, we will develop an off-line analysis tool that will allow us to interrogate the observed vehicle behavior against the simulator. This analysis will serve to tune the simulation parameters through both human intervention and a partially automated process with Monte Carlo estimation of the dozens of vehicle coefficients. Once the vehicle simulator is tuned to observed vehicle performance, we will be able to detect anomalies by highlighting deviations of the vehicle performance from expected values. Furthermore, unexpected and previously unobserved anomalies can be identified in situ by detecting when observed performance fails to conform to the simulated performance.

Our work on the compute device scheduler has taken a different course over the last year than in previous years. Our previous work on this system focused on a simple port of the Distributed OpenCL platform that we developed for desktop computer applications. With our increased focus on in-vehicle systems, our priorities have necessarily shifted. While it is possible for compute devices to fail or be shut down while deployed, the cluster is not as dynamic as one would find in an office-type environment. With that in mind, we have been working on developing the likely use cases of on-vehicle dynamic compute clustering, and refining the theory appropriately. For example, in an office environment, it is possible to add systems to the cluster at any moment, and indefinitely. As a result, systems that have never been known to the system may appear. In a vehicle, the physical equipment is fixed, and it is presumed that every device is known to the system. It is possible to modify the partitioning and configuration at any time, but no device may be physically installed or removed (other than through extraordinary measures). Therefore, it is possible to simplify and streamline parts of the dispatcher logic.

Targeting vehicle systems with the compute scheduler introduces new challenges. When we consider running vital vehicle algorithms through the dispatcher, it is extremely important to consider the

stability and reliability implications. With the compute dispatcher, it is theoretically possible to improve total system reliability, but only if we do not introduce a single point of failure. Over the past year, we have been investigating the possible solutions that would allow for an improvement in system reliability and resilience in the case of single and multiple system failures while improving performance under normal circumstances.

An unfortunate prerequisite for the deployment of the dispatcher on real-world hardware is the fragmentation of Linux support of heterogeneous embedded architectures. It is tedious, but necessary, to ensure a consistent set of userspace libraries upon which the dispatcher and client code are built. Although the dispatcher supports heterogeneous scheduling and processing, Linux is known for its sensitivity to changes in the processor architecture, especially the small differences between ARM variants. Therefore, we are developing a subset of the Linux/GNU operating system that should reduce the time required to build the kernel and userspace significantly. This part of the work is vital to ensure the modularity and agility of our systems and the vehicles it supports.

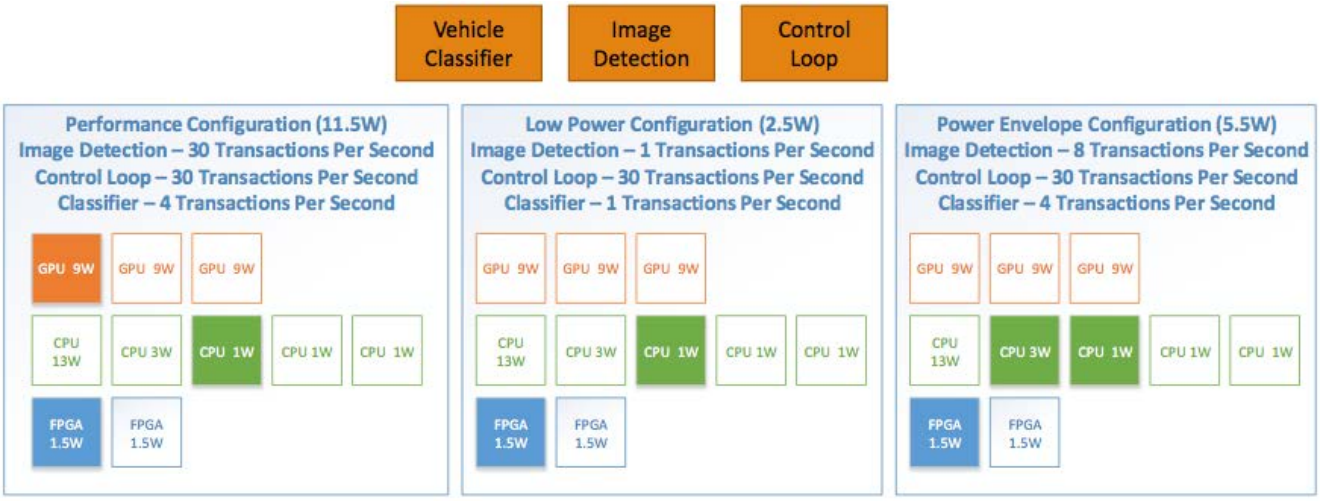

Figure 1 - Sample configurations of on-board computing elements. The shaded tiles represent the selection of some subset of available computing elements enabled at some point in time. The total system power is computed as well as the computing throughput given a sample workload.

Lastly, we have begun the foundational and theoretical work on power-aware scheduling. The current trend in microprocessor design is an increased emphasis on specialization. When systems must be adaptive, or are expected to have relatively few units manufactured, it is now common to see field-programmable gate arrays (FPGAs) added to designs, sometimes replacing general-purpose processors. In cases where a large number of units are manufactured and functionality isn't expected to change, it is common for parts of the system to be implemented in raw silicon. In the recent past, there was a movement toward general purpose processors and away from custom application-specific integrated circuits (ASICs). The computing industry is often cyclical, and this is an example of such a shift. To achieve the maximum performance in terms of operations per watt, it is necessary to be both adaptive to dynamic workloads and able to make use of the most recent developments in computing technology. Our work in the past year has been to develop the theoretical underpinnings of power-aware scheduling, but to also understand the nature of the computing devices that are to come and to be sure that our work will continue to support their use. As part of that exploration, we modeled scenarios with

differing mixes of computing devices, and we are trying to characterize the performance implications of those combinations (Figure 1 is an example of such a characterization).

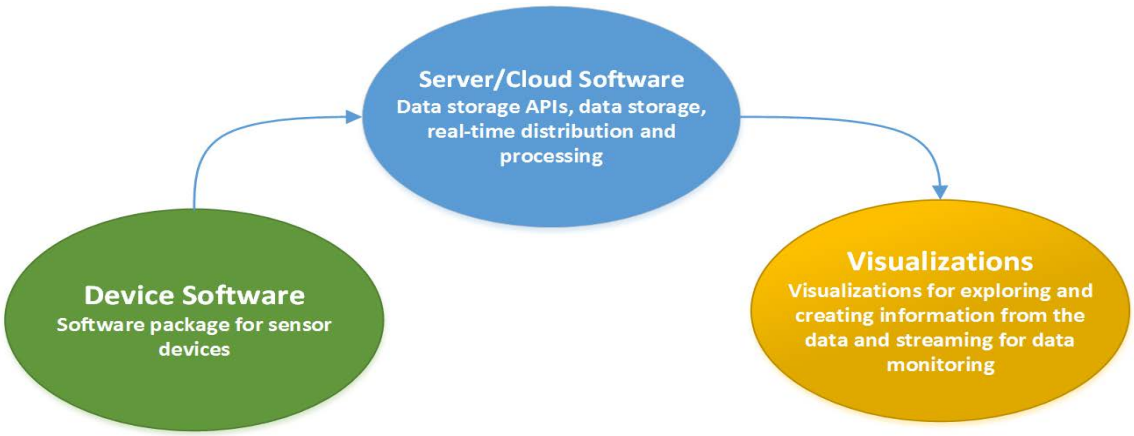

Figure 2 - SensorStream architecture

SensorStream is a cloud service for time-series data storage and distribution, built from industry-standard protocols. All of the interactions with this system are through stable standards-compliant web APIs using well established protocols. The protocols used for communication are JSON, REST, and HTTP(s), each of which has a large number of tested and well-documented open source libraries, some with unencumbered licenses. This allows for rapid development by leveraging existing tools. This system is also compatible with commodity cloud providers such as Azure, EC2, and Rackspace, as well as private infrastructure. This is allowing the platform to be used not only as a service, but also as a private deployable package. As much as possible, this system was built using operating system-agnostic protocols and libraries in order to ensure compatibility with as many users as possible.

The SensorStream architecture is highly scalable, capable of being run on just a few small systems, or scaling up to thousands of large systems. The backend database is built on Cassandra, which is a linearly scalable and fault-tolerant data store that can be run on commodity hardware or cloud infrastructure. It is used by many large scale web companies such as Netflix and capable of scaling to store petabytes of data and include one hundred thousand nodes. The web API was developed using .NET web handlers running on Microsoft Internet Information Server (IIS). The API, data distribution, and backend storage were all developed as separate pieces to allow for modularity, and created with scalability in mind.

The full featured web API allows for posting, querying, and retrieving measurements or any other type of time-series data. There is also a messaging bus for subscribing to data being inserted into the system. This was built using a SignalR library, which uses HTTP long polling, or WebSockets, depending on the capabilities of the library or browser. In addition to the open protocols and publicly accessible APIs, we developed several libraries for client access. The suite of libraries currently support Objective-C, Ruby, C#.NET, JavaScript, and C and are capable of running on Apple iOS, Macintosh OSX, Android, Microsoft Windows, Linux and any modern web browser, independent of operating system.

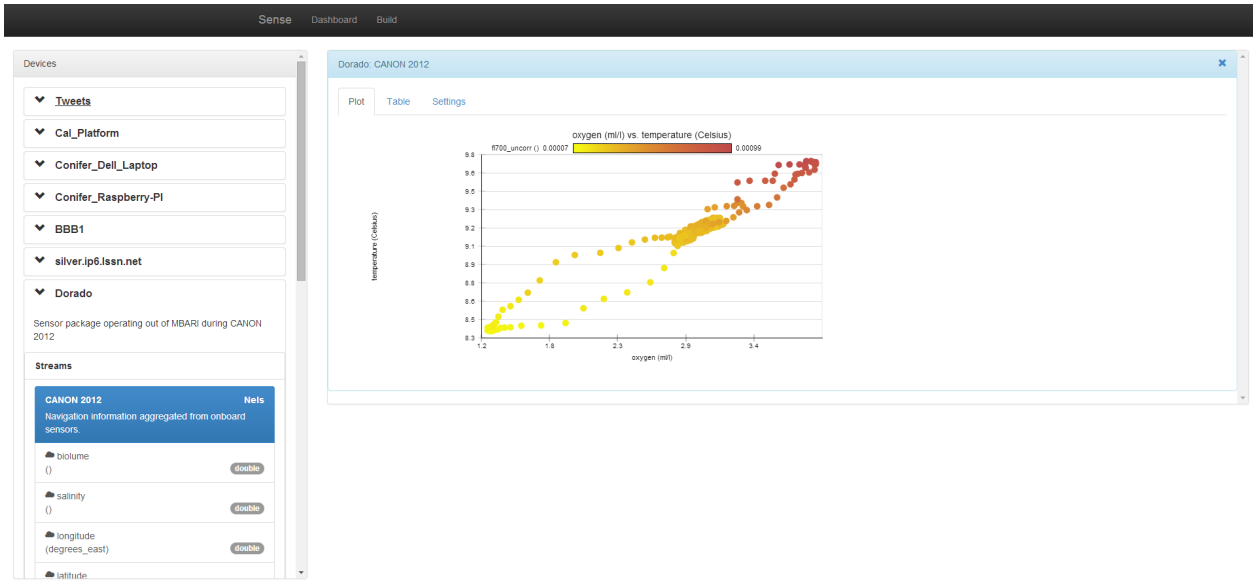

Figure 3 - Example graph from the SensorStream web portal

In order to make this service more useful we created a front end visualization and graphing portal. Implemented in JavaScript and hosted on a public website to allow the largest number of people to access the service from a wide variety of devices. This site includes the ability to retrieve data in tabular and graph form. Users can create one to one relations between different streams of data (i.e. comparing temperature to time, or temperature to elevation) helping them to make sense of the collected data. This site also implements streaming data, allowing users to create real-time dashboards and monitor their deployed sensor platforms.

RESULTS

The major results of our work in failure detection and power aware scheduling have yet to be fully realized. We are a few months into several long-term explorations, and cannot making assertions about this in-progress research. However, the SensorStream project is at a point in its evolution that is stable enough to address results and lessons learned. As mentioned in the work completed section, we leveraged the state of the art in cloud service back-ends. It became clear to us early in the work that the database solution was going to be the most important piece to get right. In an application such as this, a traditional standard query language (SQL) database would limit the scalability long term which is why we selected Cassandra. The web layer that sits above the database would have to be parallel such that multiple threads across multiple servers could access the database in a high-performing and consistent way. The throughput we are currently achieving from a single client is on the order of 300 transactions a second, and that is limited by the client itself. The SensorStream backend can process a transaction within one millisecond, on average, per thread. Outside of the main processing flow, statistical summaries are generated for every stream for every hour, and those are computed in 14 milliseconds on average. Finally, we have found that system modularity is extremely important. Maintaining the modularity of the components of the SensorStream system allows us be agile while tailoring the platform to new workloads, system services, and infrastructure environments.

IMPACT/APPLICATIONS

The work on power-aware scheduling may significantly impact the performance of autonomous vehicles. The rapid application of the newest processors could both increase processing throughput and decrease the power envelope. This work would also support the scaling of available compute resources as appropriate for the task at hand. A prime concern for vehicle control algorithm designers is the time to solution. The world does not stop while the planner is computing, and there is a real risk that the computed plan is only valid for a moment that has already passed. However, it may not make sense to support an abundance of computing capability that is idle the majority of the time. Our system could enable rapid computation of time-sensitive problems without sacrificing the average power budget.

Our work on SensorStream has already demonstrated an impact in real-world applications. Though it has been in development for only a year, the platform is so useful that we are routinely asked to provide it to others, which we have not done. SensorStream hits the right balance between sophisticated real-time publication and subscription of sensor data and simple deployment. Our work with this system is ongoing, and we expect to add a management portal that would allow users to manage the devices that they have provisioned, add more sophisticated and interesting graph and data analysis, and provide a mechanism for "data dashboards" that are a curated collection of graphs from several streams that provide a consistently useful picture into some process or system.

On the subject of fault and anomaly detection in autonomous vehicles, this work has the potential to dramatically improve the mean time between failure, or mean time between intervention. As the size of autonomous vehicle fleets grow, it becomes impractical to maintain the level of human supervision that is currently required. Even the state of the art work on failure and anomaly detection requires accurate and complete classifier coefficients. There are no automatic tools for the discovery of these coefficients in this domain, and perhaps not even outside of this domain. This work has the potential to dramatically improve the space of anomaly detection.

RELATED PROJECTS

None