KDE is a non-parametric model that does not assume any particular structure for the target distribution (linear, quadratic, etc.). It uses a historical data set to construct a conditional probability density function to make estimates. The density estimate is similar to a histogram. On each data point, we put a probability mass and then sum all the point masses to get the joint density estimate:

$$ f(x, y) = \frac{1}{N} \sum_{i=1}^{N} K(x - x_i)K(y - y_i) $$

We can then calculate the conditional expectation of $y$ given $x$:

$$ E[y|X] = \int y f(x, y) dx = \frac{\sum_{i=1}^{N} y_i K(x - x_i) }{ \sum_{i=1}^{N} K(x - x_i) } $$

There are several nice things about the form of this expression. First, we can use any sort of variable for the predictors. Wind speed, wind direction, temperature, time of day, day of year can all be predictor variables. These variables can come from multiple sites, including the site where the estimate is being made, neighboring measurement sites, and grid points from a numerical weather forecast such as the NAM 12km model. It is because of this flexibility that we can use this approach on different application types, such as forecasting and site assessment. Also note that during parallelization the algorithm can be broken down into relatively independent pieces to minimize the communications burden on a distributed memory machine.

**Forecast Analysis on Wind Speeds at MIT**

To evaluate the effectiveness of our methodology, we analyzed a test site on MIT campus. Data was taken from sensors on the top of the Green Building (Building 44) on the east side of campus. There are plans to install a small-scale turbine on campus by the end of 2010. The turbine installation is being planned by MIT Facilities and the MIT Full Breeze student group.

**Computation and Results**

For this experiment, we used outputs from the NAM 12km Model to make forecasts at a location on MIT campus (see lower left). We used hourly data for the year 2009 to make one hour ahead predictions and compared the performance of our kernel density estimates against persistence and linear regression.

The kernel regression estimate performed better on average than both of the other techniques. Kernel regression had a MSE 40% lower than persistence and 12.5% percent lower than linear regression.

The second graph shows how tuning the "knobs" of the algorithm allows the user to trade accuracy for faster computation. The more predictor variables used, the higher the accuracy achieved, but at a higher computational cost.

These results were obtained using a relatively small set of predictor variables. We hope to achieve better results with more room for improvement using a better estimation algorithm (discussed below) and more diverse predictor variables.

We also ran a measure-correlate-predict (MCP) analysis on NOAA ocean buoy data. The kernel density estimation achieved an improvement of greater than 25% (in terms of mean squared error versus observed) compared to using the variance-ratio MCP method in estimating missing historical data. These MCP computations were performed on a SGI Altix 330 machine with 12 Intel Itanium 2 processors running Interactive Supercomputing’s STAR-P software. Overall, using all 12 processors on the machine, a 8.9x speedup compared to serial performance was achieved.

In the future, we plan to use a modification of the algorithm presented here to minimize the mean squared error plus a regularization term, which helps with generalization. This estimation algorithm is more complex as it involves solving a large symmetric linear system to attain the objective minimization. Moving to this more complex algorithm will provide greater accuracy as well as fertile ground for exploring performance vs. accuracy trade-offs.

**Conclusions**

We have shown that the use of tunable kernel density estimation and regression techniques can be applied effectively when leveraging high performance parallel computing resources. Not only are the results achieved better than those produced when using methods such as persistence and linear regression, but also the algorithms are tunable to allow the user to trade accuracy for computational performance and vice versa to suit the user’s needs.

These types of techniques will become ever more important as parallel computing becomes ubiquitous across all types of computing platforms. As software developers struggle to update their programming practices to utilize these types of resources, techniques such as the automatic tuning of performance parameters to achieve the user’s desired results will become extremely valuable. In the future, we plan to implement these techniques with the PetaBricks programming language, which will do automatic algorithm selection and parameter tuning to achieve high performance, portable, parallel, variable accuracy software for wind prediction applications.