FPGA-Accelerated Isotope Pattern Calculator for Use in Simulated Mass Spectrometry Peptide and Protein Chemistry

Carlo Pascoe, David Box, Herman Lam, Alan George
NSF Center for High-Performance Reconfigurable Computing (CHREC)
Dept. of Electrical and Computer Engineering, University of Florida
Gainesville FL, USA
E-mail: {pascoe, box, hlam, george}@chrec.org

Abstract— **Over the past 20 to 30 years, the analysis of tandem mass spectrometry data generated from protein fragments has become the dominant method for the identification and classification of unknown protein samples. With wide ranging application in numerous scientific disciplines such as pharmaceutical research, cancer diagnostics, and bacterial identification, the need for accurate protein identification remains important, and the ability to produce more accurate identifications at faster rates would be of great benefit to society as a whole. As a key step towards improving the speed, and thus achievable accuracy, of protein identification algorithms, this paper presents a FPGA-based solution that considerably accelerates the Isotope Pattern Calculator, a computationally intense subroutine common in de novo protein identification. Although previous work shows incremental progress in the acceleration of software-based IPC (mainly by sacrificing accuracy for speed), to the best of our knowledge this is the first work to consider IPC on FPGAs. In this paper, we describe the design and implementation of an efficient and configurable IPC kernel. The described design provides 23 customization parameters allowing for general use within many protein identification algorithms. We discuss several parameter tradeoffs and demonstrate experimentally their effect on performance when comparing execution of optimized IPC software with various configurations of our hardware IPC solution; we demonstrate between 72 and 566 speedup on a single Stratix IV E530 FPGA. Finally, a favorable IPC configuration is scaled to multiple FPGAs, where a best-case speedup of 3340 on 16 FPGAs is observed when experimentally evaluated on a single node of Novo-G, the reconfigurable supercomputer in the NSF CHREC Center at Florida.**

Keywords-Isotope pattern calculator; Mass spectrometry; Protein identification; FPGA acceleracion; Reconfigurable computing

I. INTRODUCTION

Proteins are biochemical molecules consisting of one or more polypeptides, where polypeptide is a macromolecular chain of linked amino acids. Over the past 20 to 30 years, the analysis of tandem mass spectrometry data generated from polypeptide fragments has become the dominant method for the identification and classification of unknown protein samples. With wide-ranging application in numerous scientific disciplines such as pharmaceutical research [1], cancer diagnostics [2], and bacterial identification [3], the need for accurate protein identification remains important and the ability to produce more accurate identifications at faster rates would be of great benefit to society as a whole. Unfortunately, as detailed in [4] and [5], current industry-standard methods such as Mascot [6], SEQUEST [7], and PEAKS [8] remain unreliable at best, further stressing the need for new solutions to this challenging problem. As a key step towards improving the speed, and thus achievable accuracy, of protein identification algorithms, this paper presents a FPGA-based solution that considerably accelerates the Isotope Pattern Calculator (IPC), a computationally intense subroutine common in *de novo* protein identification.

In mass spectrometry, the bonds that hold together an *unknown* protein's amino acid chain are systematically broken and the abundance of protein fragments at each particular mass to charge ratio (m/z) are measured. Figure 1a shows sample mass spec. output where each peak represents the relative abundance of fragments at that particular m/z. Protein identification algorithms interpret this *experimental spectrum* and attempt to generate an amino acid string representing the primary structure of the unknown protein. By way of database comparison or theoretical prediction (i.e., de novo), algorithms perform an iterative reduction of potential protein search space until a best guess as to the identity of protein(s) in the sample are produced. The more widely used database methods rely on the ability to search extensive collections of previously identified protein spectrum data, with correct solutions contingent upon *preexistence in a database*. The popularity of database methods over de novo can be attributed to the exponential increase in available sequence data, relative simplicity of effective algorithms, and ability to converge on solutions orders of magnitude faster than de novo methods with comparable accuracy [9].

However, when database methods are not appropriate, the less popular de novo methods are used to generate *theoretical mass spectra* for each candidate sequence and by comparing it to the experimental spectrum, candidate quality is iteratively improved until theoretical and experimental spectra match. The relative computational inefficiencies of de novo methods stem from the theoretical need to consider all possible linear combinations of amino acids as potential candidates. The number of candidate sequences grows exponentially with the final sequence length, potentially requiring intractable computation even for moderately sized peptides, let alone large proteins! Exchanging accuracy for speed, different algorithms employ diverse heuristic pruning methods to limit the potential computational intractability, a necessity for practical use on conventional computing systems that often leads to false identifications. By

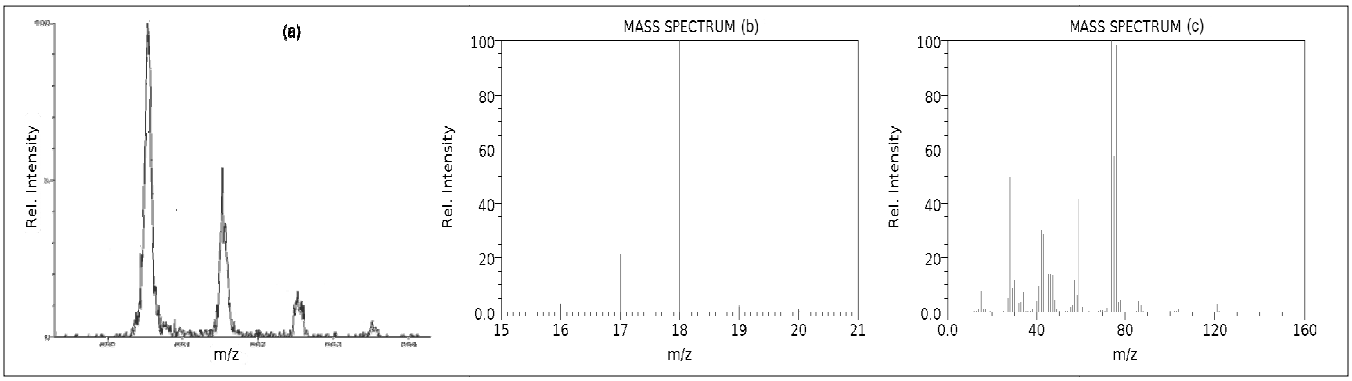

Figure 1. (a) Sample mass spectrum, (b) Theoretical isotopic distribution for a single molecule of water, H_2O, and (c) Theoretical isotopic distribution for the single amino acid Cysteine, $C_3H_7NO_2S$. Note that peaks with less than 0.1% relative intensity are not shown.

accelerating key computation common in many de novo algorithms, algorithm developers can employ less restrictive pruning criteria, potentially allowing a greater degree of accuracy in less time.

Profiling indicates the majority of execution time for many highly accurate de novo algorithms involves the calculation of theoretical mass spectra for each candidate sequence. This calculation comprises the methodical decomposition of a candidate sequence string into many amino acid substrings representing possible protein fragments that may have contributed to the experimental spectrum, the calculation of probable mass contributions for each predicted fragment, and finally the histogram-like combination of probable masses to form a theoretical distribution directly comparable to experimental mass spectra. This seemingly straightforward calculation is greatly complicated by the fact that, in nature, elements occur as a mixture of isotopes, where atoms of a given element contain a constant number of protons, but differ in the number of neutrons. Potential neutron quantity differences of constituent atoms within molecules of the same chemical formula suggest molecules have a distribution of possible masses rather than a single value. To calculate the theoretical isotopic mass distributions of fragment substrings, chemist use the previously mentioned IPC subroutine which enumerates all possible combinations of constituent element isotopes and produces a list of mass/probability pairs, each pair corresponding to a possible mass for a molecule of the given chemical formula and its probability of occurring. Although a relatively simple calculation for the smallest of molecules, IPC executions for medium- to large- sized molecules quickly become a computational bottleneck of many chemistry applications, most notably de novo protein identification. Figure 1b shows example IPC output for a relatively tiny fragment, while Figure 1c shows output for a larger yet still small fragment. Recognizing that each peak results from many addition, multiplication, division, and exponentiation operations, visually comparing the two figures gives insight into the potential for excessive computation in a single IPC calculation as fragment size increases. Recognizing that each predicted fragment of each candidate sequence (the number of which grows exponentially with protein length) requires its own IPC calculation gives insight into the potential for computational intractability, especially when utilizing the more accurate algorithms that explore a larger portion of total protein search space.

Based on computational intensity and perceived execution dominance, IPC is chosen as the logical beginning to accelerate underutilized de novo methods while not sacrificing accuracy for speed. Rather than relying solely on software techniques, we feature FPGA-based reconfigurable computing and its inherent advantages in efficient, parallel execution of well-designed, application-specific hardware kernels. Although previous work [10−17] shows incremental progress in the acceleration of software-based IPC calculation, to the best of our knowledge this is the first work to consider IPC on FPGAs.

In this paper, we describe the design and implementation of an efficient and adaptable IPC kernel and its considerable impact on achieved performance. In Section II, the background and related work for IPC are given, followed by the IPC problem formulation in Section III. Although many de novo algorithms rely on IPC calculation, each one has slightly different expectations as to the provided functionality, and as such, a useful design must provide high customization and parameterization to accommodate as many algorithms and applications as possible. In Section IV, a configurable and scalable IPC hardware architecture is described. The described design provides 23 customization parameters ranging from the integer and fractional precision of internal calculations to the way in which output values are sorted and delivered to the host. Each parameter represents a tradeoff between performance and scientific requirements of the targeted mass spectrometry algorithm. In Section V, we discuss several of the key tradeoffs and demonstrate experimentally their effect on performance when comparing execution of optimized IPC software run on a single Xeon E5620 CPU core with various configurations of our hardware IPC solution run on a single Stratix IV E530 FPGA from Altera with an E5620 core for host support. Also in Section V, a favorable IPC hardware configuration is scaled to multiple FPGAs (up to 16 E530s) and experimentally evaluated on a single node of Novo-G, the reconfigurable supercomputer in the NSF CHREC Center at the University of Florida [18, 19]. Finally, Section VI provides a summary, conclusions, and planned future work.

II. Background and Related Work

This section provides limited background in the areas of mass spectrometry and protein identification, and then discusses previous work in the area of software-based IPC.

For a more in-depth review, please consult the many surveys or research articles published in these areas (e.g., [2, 9, 20]).

A. Tandem Mass Spectrometry

In order to generate tandem mass spectrometry (MS/MS) data, a potential combination of enzymatic digestion and/or collision-induced dissociation is used to break apart the linked amino acid chains of polypeptides into predictably small, electrically charged fragments referred to as peptides. By way of finely tuned magnetic fields, only fragments with a precise kinetic energy and mass to charge ratio (m/z) are permitted to travel through the mass spectrometer at a given instant to the detector where the abundance relative to other m/z fragments are measured. Peptides are then further fragmented to the amino acid level and undergo a second run through the spectrometer that produces an output of mass/abundance pairs similar to the output presented in Figure 1a.

The first uses of MS/MS for protein sequence analyses date back to the 1980s [9]. Although there have been many revisions to methods and technologies used in MS/MS over the years, the main objective of obtaining the most accurate mass readings of individual fragment ions has remained the same. Recent research has navigated towards the so-called "hybrid" mass spectrometers able to switch between several different scan modes [21] and different collision energies [22] to better identify individual peptides as the final resolution and accuracy of the output is heavily dependent on the instrument and fragmentation method used.

B. Protein Identification

At the time when mass spectrometry was first proposed as a tool for protein identification, the method of choice consisted of the repeated removal and identification of single amino acids from peptide structures. This procedure had several limitations, most notable the prohibitively slow speed and the requirement of extremely pure test samples. In 1980, [23] proposed the combined use of this method with field-desorption mass spectrometry, starting the trend that has led to analysis of MS/MS data as the dominant method used today. With MS/MS data, protein identification algorithms attempt to generate an amino acid sequence representing the whole protein by sequencing each constituent fragment and piecing them together. The rational behind this bottom-up approach is that each peptide or peptide fragment possesses a more easily discernible, unique mass signature (with slight variance based on isotopic distribution) that acts much like a fingerprint. If the protein under study shows many of the same fragments in its mass spectrum as experimentally proven or predicted fragments of some known protein, there is a high likelihood the two proteins are the same or at least similar. Each algorithm performs an iterative reduction of the potential protein search space until a best guess as to the identity of protein(s) in the sample are generated. Based on many factors such as poor quality mass spectrometry data, non-ideal peptide fragmentation, or poor quality of identification algorithms, generated sequences may or may not be correct.

As mentioned in Section I, protein identification algorithms historically fall into the classifications of database or de novo, with many modern algorithms actually a combination of the two. Different database methods can be non-probabilistic, probabilistic, and/or heuristic approaches. Non-probabilistic database methods, such as SEQUEST [7], use a cross-correlation function (or similar) to match database information to measured ion mass data. MASCOT [6], currently one of the most popular database identification algorithms, uses a probability-based scoring algorithm to rate the quality of experimental data to database data. Probability-based techniques are employed to speed up database searches, especially when paired with a stochastic searching model. Heuristic approaches, such as DBDigger [24], are able to show drastic speedup over other optimal database algorithms by lowering data dimensionality, limiting the size of searched databases, and pruning away large portions of the potential protein search space. As expected, these heuristics lead to less accurate results, often times resulting in false identifications.

De novo methods such as SHERENGA [25] and PepNovo [26] do not rely on databases, but rather predict the best possible sequence exclusively from a given experimental spectrum. By generating theoretical mass spec. data for each candidate sequence and comparing it to the experimental spectrum, candidate quality is iteratively improved until theoretical and experimental spectra match. The relative computational inefficiencies of de novo methods stem from the theoretical need to consider all possible linear combinations of amino acids as potential candidates; the number of candidate sequences grows exponentially with the final sequence length, potentially requiring an intractable amount of computation even for small peptides, as seen from the de novo benchmarks in [25] and [26].

Increasingly, separation between database and de novo methods has narrowed as hybrid methods employ both de novo and database methods where appropriate on sub-portions of input data. In order to reduce database sizes, and therefore search time, the de novo sequencing approach has recently been used to generate tags for database filtration [27, 28]. This technique, sometimes called precision mass spectrometry, allows applications to forgo the time-consuming step of matching each query spectrum to every database peptide. These hybrid approaches can reduce the total runtime of protein identification by two orders of magnitude [28] for larger proteins.

C. Isotopic Pattern Calculators

The most popular isotope pattern calculation algorithms use one of two primary approaches, polynomial expansion [10, 11] or Fourier transform [12, 13]. Both methods are mathematically related through the polynomial form of the IPC problem definition [14]. Polynomial expansion methods use a stepwise approach that obtain a theoretically perfect resolution, but suffer from slow execution and limited memory. To lessen these issues, several algorithms [11, 15, 16] apply a probability threshold by which low probability data is pruned from final output. Other algorithms [17] allow the user to set the word width allocated for storing mass and probability variables, allowing the user to reduce precision, and thus memory footprint.

Applying convolution to the IPC polynomial, several algorithms attempt to accelerate the IPC calculation by using the Fast Fourier Transform (FFT) [12−14]. The relative speedup of FFT-based methods is at the expense of reduced resolution [14].

III. IPC Problem Formulation

As indicated in Section I, the Isotope Pattern Calculator enumerates all possible combinations of constituent element isotopes within a given chemical formula and produces a list of mass/probability pairs, each pair henceforth referred to as a "peak." The full IPC calculation is analogous to evaluating a polynomial of the form

$$(E_1^1 + E_2^1 + \cdots)^{N_1}(E_1^2 + E_2^2 + \cdots)^{N_2}(E_1^3 + E_2^3 + \cdots)^{N_3}\cdots,$$

where E_i^j represents the i^{th} isotope of the j^{th} unique element in a chemical formula containing N_j atoms of the j^{th} element type [11]; each resulting term represents a possible peak, calculated by adding the masses and multiplying the probabilities of elemental isotopes in a term. This calculation can be broken down into a two-stage process with stage 1 focusing on the calculation of single-element distributions (SEDs) for each constituent element and stage 2 the iterative combination of each SED.

A. Stage 1: Calculation of sigle-element distributions

Consider the element Hydrogen with two stable isotopes: ${}_1^1H$ with a mass of 1.0078 Da occurring 99.99% in nature and ${}_1^2H$ with a mass of 2.0141 Da occurring 0.01% in nature. If the input chemical formula contains a single Hydrogen, H_1, no stage 1 calculation for Hydrogen is required and its SED is simply the two peaks

{ (1.0078, 99.9885%), (2.0141, 0.0115%) }

corresponding to ${}_1^1H$ or ${}_1^2H$. However, if the chemical formula contains more than a single atom of the same element, for instance H_2, determining the distribution becomes a combinatorics problem; by enumerating all possible *order-independent*, 2-combinations from the set of two Hydrogen isotopes (i.e., ${}_1^1H$ ${}_1^1H$ or ${}_1^1H$ ${}_1^2H$ or ${}_1^2H$ ${}_1^2H$), then adding their masses and performing a binomial expansion on their probabilities, the H_2 SED becomes

{ (2.016, 99.977%), (3.022, 0.022%), (4.028, 1.32e^{-6}%) }.

A general procedure for calculating the distribution for an arbitrary number of Hydrogen atoms, H_N, is provided as Algorithm 1. Note that the amount of computation increases as the number of atoms increase.

The procedure provided in Algorithm 1 is easily adapted to handle any element with two stable isotopes (e.g., Helium, Carbon, Nitrogen, etc.) by substituting the appropriate probabilities and masses, but for those elements with 3, 4, all the way up to 10 stable isotopes for Tin, a different procedure is required. Rather than enumerate all combinations from a set of two possibilities, a set of k possibilities is required, where k is the number of stable isotopes. Of the five naturally occurring elements in polypeptide structures (Hydrogen, Carbon, Nitrogen, Oxygen, and Sulfur), Sulfur has the most stable isotopes with four. A general procedure for calculating the distribution for an arbitrary number of Sulfur atoms, S_N, is provided as Algorithm 2. Much like Algorithm 1, this second algorithm is easily adapted to handle any element with four stable isotopes (e.g., Lead, Iron, etc.). Note that the amount of computation significantly increases as the number of stable isotopes increase. In general, for an element with k stable isotopes, multinomial expansion within $(k-1)$ nested FOR loops is required.

ALGORITHM 1. Calculate single-element distribution for H_N

1: $p_0 \leftarrow 0.999885$, $m_0 \leftarrow 1.007825$
2: $p_1 \leftarrow 0.000115$, $m_1 \leftarrow 2.014102$
3: ***FOR*** $n_1 \leftarrow 0$ to N
4: $n_0 \leftarrow \mathrm{N} - n_1$
5: $p \leftarrow \binom{N}{n_1} p_0^{n_0} p_1^{n_1} = \frac{N!}{n_1! n_0!} p_0^{n_0} p_1^{n_1}$
6: $m \leftarrow n_0 m_0 + n_1 m_1$
7: ***PRINT*** (m, p)
8: ***END LOOP***

B. Stage 2: Iterative combination for final distribution

The stage 1 output is equivalent to IPC output in the event that a chemical formula contains only a single element type. For molecules with multiple element types, individual peaks from the SEDs must be combined in a pairwise fashion to form the final distribution. This calculation is analogous to evaluating a polynomial of the form

$$(P_1^1 + P_2^1 + \cdots)(P_1^2 + P_2^2 + \cdots)(P_1^3 + P_2^3 + \cdots)\cdots,$$

where P_i^j represents the i^{th} peak from the stage 1 SED generated for the j^{th} unique element.

Consider the stage 2 calculation for H_2O. Stage 1 produces three peaks for Hydrogen, as previously shown, and three peaks for Oxygen, its 3 stable isotopes. Stage 2 combines the two distributions to produce the final nine peak distribution (displayed graphically in Figure 1b). The relatively small H_2O molecule requires a relatively small amount of computation. Now consider calculation for the *single* amino acid Cysteine with a chemical formula of $C_3H_7NO_2S$ which produces 1152 unique peaks (Figure 1c). Though $C_3H_7NO_2S$ is still relatively small, there is considerably more computation. Noting that typical protein fragments encountered in mass spectrometry range anywhere from a few to over 50 amino acids, computation can quickly become prohibitive for conventional computing systems.

C. Additional IPC functionality

Often times the exact theoretical distribution provides many low-probability peaks with negligible effect on higher-level algorithm output. The use of a threshold probability allows filtering of peak calculations that fall below the threshold, reducing frivolous calculation and lowering the number of reported peaks. Imposition of a threshold probability is by far one of the most widely used IPC pruning techniques.

ALGORITHM 2. Calculate single-element distribution for S_N

1: $p_0 \leftarrow 0.9493$, $m_0 \leftarrow 31.972079$
2: $p_1 \leftarrow 0.0076$, $m_1 \leftarrow 32.971459$
3: $p_2 \leftarrow 0.0429$, $m_2 \leftarrow 33.967867$
4: $p_3 \leftarrow 0.0002$, $m_3 \leftarrow 35.967081$
5: ***FOR*** $n_1 \leftarrow 0$ to N
6: ***FOR*** $n_2 \leftarrow 0$ to $\mathrm{N} - n_1$
7: ***FOR*** $n_3 \leftarrow 0$ to $\mathrm{N} - (n_1 + n_2)$
8: $n_0 \leftarrow \mathrm{N} - (n_1 + n_2 + n_3)$
9: $p \leftarrow \binom{N}{n_1}\binom{N-n_1}{n_2}\binom{N-n_1-n_2}{n_3} p_0^{n_0} p_1^{n_1} p_2^{n_2} p_3^{n_3}$
10: $= \frac{N!}{n_3! n_2! n_1! n_0!} p_0^{n_0} p_1^{n_1} p_2^{n_2} p_3^{n_3}$
11: $m \leftarrow n_0 m_0 + n_1 m_1 + n_2 m_2 + n_3 m_3$
12: ***PRINT*** (m, p)
13: ***END LOOP***
14: ***END LOOP***
15: ***END LOOP***

Although IPC output graphically represents a spectrum of mass peaks, the internal manifestation as a list of mass/probability pairs allows for several different pair orderings representing the same information. Based on the intended use by higher-level algorithms, it may be advantageous to sort the peaks, either by mass or by probability. Sorting by mass allows for easy graphing and peak merging. Sorting by probability allows for quick analysis of the most probable peaks. If an algorithm has no preference for pair ordering it is more advantageous to skip sorting altogether and report peaks in the order in which they are produced, significantly speeding up computation.

Some algorithms only handle output with a maximum number of peaks, requiring output with more peaks than a given number *N* to be trimmed [11]. If output is sorted by probability, then the *N* most probable peaks are kept and the rest discarded; this has the effect of presenting a coarse-grained view of the entire spectrum. If output is sorted by mass, then *N* consecutive peaks within a range of masses are kept, regardless of probability; this has the effect of presenting a fine-grained view of the entire spectrum within a limited *mass-window*.

As mentioned in [2], some algorithms such as [8] attempt to reduce total calculation during distribution data preprocessing by first performing *peak centroiding*. In order to reduce the dimensionality of fine-grained distributions, adjacent peaks within a tiny mass-window (fractions of a Da) are merged into a single peak by way of weighted average. As seen in Figure 1c, distributions naturally form several clusters of close peaks separated by ~1Da (approximate mass of a single Neutron). Centroiding generally has the effect of reducing distributions to a list of single peaks separated by about a single Da. Assuming peaks are sorted by mass, centroiding is easily performed after IPC with a sequential pass through distribution output.

IV. A Configurable and Scalable IPC Hardware Architecture

In this section, we describe an adaptation of the two-stage process described in Section III to a configurable and scalable hardware architecture capable of converting a stream of independent chemical formula queries (not to be confused with candidate protein queries from which the formula queries are generated) into a delimited stream of variable-quantity mass/probability peaks. Figure 2 provides a high-level block diagram for the general IPC hardware architecture. The design largely consists of two hardware modules, one responsible for stage 1 (Figure 3) and the other responsible for stage 2 (Figure 4).

A. Stage 1: Calculation of SEDs reduced to lookup tables

Theoretically, possible queries comprise an unbounded number of atoms from all possible elements. Practically, when considering peptide fragments generated in mass spectrometry, queries almost never possess more than a few hundred atoms from the set of five naturally occurring elements in polypeptide structures (H, C, N, O, & S). If other elements are present in a peptide fragment, usually introduced by the chemical processes used in fragmentation, their atom quantity is usually no more than one or two.

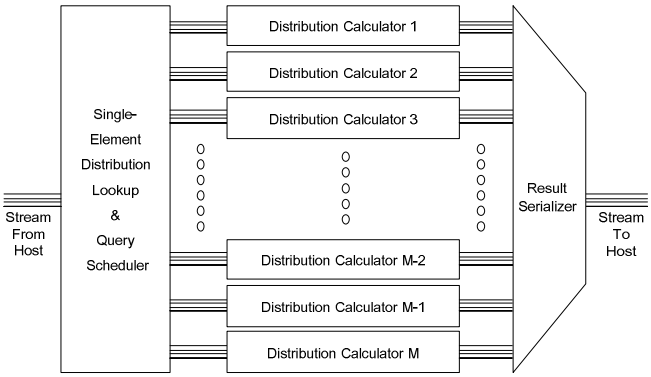

Figure 2. High-level block diagram for IPC hardware design; left most block performs IPC stage 1; calculator blocks performe stage 2.

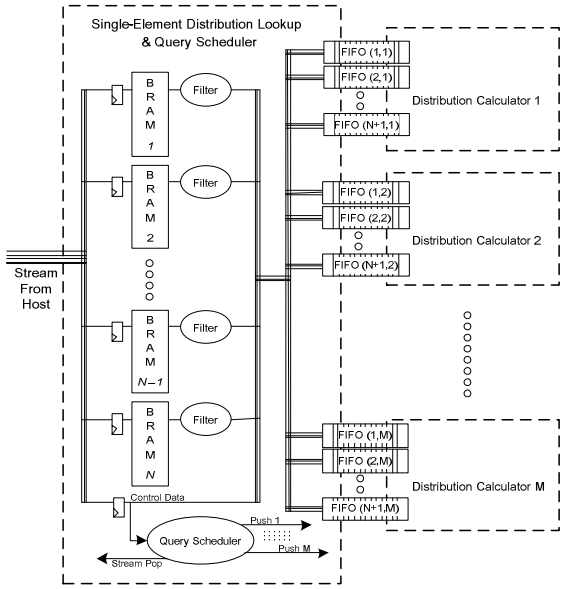

Figure 3. High-level schematic for IPC "single-element distribution lookup & query scheduler" block.

Precomputed SEDs in LUTs. Recognizing an overwhelming majority of encountered IPC queries will fall within a relatively-small finite subset (e.g., all chemical formulae with up to 256 H, C, N, and O, up to 64 S, and up to 16 of 60 other elements), we eliminate the need for run-time stage 1 execution in hardware by precomputing SEDs and storing them in a bank of N *lookup tables* (LUTs), where *N* is configurable. In the rare event queries are from outside the subset, run-time computation can take place in software. Each LUT address corresponds to a unique SED (e.g., H_{233} or S_9). Relying on SED lookup rather than computation provides ***enormous performance benefits*** in terms of both reduced execution time and resource usage, all without an accompanying penalty in accuracy. Precomputation of SEDs is performed exactly (i.e., without low-probability peak filtering) then sorted by probability. LUT 1 holds the most probable peak and LUT *N* holds the N^{th} most probable. When a SED contains *X* peaks, where $X < N$, LUTs X+1 to *N* hold zero probability peak data. When $X > N$, the least probable $X-N$ peaks are omitted from stage 2 calculations. A configurable threshold probability (TP) filter is applied to each LUT output that allows sufficiently probable peaks to

ALGORITHM 3. Distribution Calculator Procedure

```
1:  WHILE Control ≠ "done"
2:    SED[1…N]←FIFO[1…N].pop(), Control←FIFO[N+1].pop()
3:    IF Control = "begin"
4:      PrevItBuff[1…N]←SED[1…N], PrevItBuff[N+1…Y]←(-1,0)
5:      CurrItBuff[1…Y] ←(-1,0)
6:    IF Control = "middle" or Control = "end"
7:      WHILE tmp←PrevItBuff[1..Y].shift() ≠ (-1,0)
8:        i← 1
9:        WHILE i≤ N and SED[i].prob > 0
10:         MultAdd[1…X].mass←SED[i…i+X−1].mass + tmp.mass
11:         MultAdd[1…X].prob←SED[i…i+X−1].prob ∗ tmp.prob
12:         PSort[1…X]←Sort(Filter(MultAdd[1…X], TP))
13:         CurrItBuff[1…Y]←InSort(CurrItBuff[1…Y], PSort[1..X])
14:         i← i + X
15:       END LOOP
16:     END LOOP
17:     PrevItBuff[1…Y]←CurrItBuff[1…Y]
18:     CurrItBuff[1…Y] ←(-1,0)
19:   IF Control = "end"
20:     FinalResBuff[1…Y]←PrevItBuff[1…Y]
21: END LOOP
```

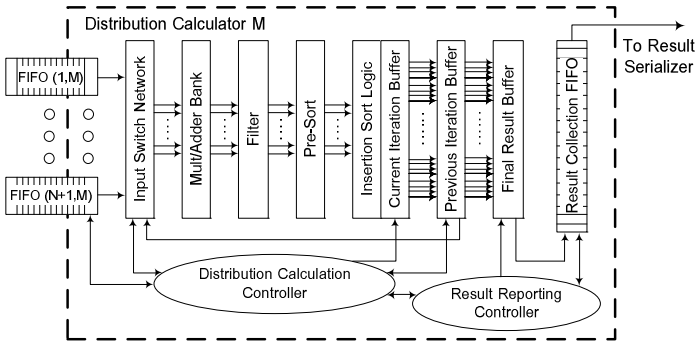

Figure 4. High-level diagram for an instance of IPC "Distribution Calulator" block.

pass but zeroes out those with low probabilities. By applying TP this way in hardware, as opposed to the time of LUT data generation, allows for run-time TP tuning. Limiting the number of non-zero peaks input to stage 2, either by lowering *N* or by increasing TP, can greatly reduce computation time at the expense of IPC output exactness.

In-stream control. Using a slightly altered form of the *in-stream control* methodology described in [29], queries are delivered to hardware by way of streamed 16-bit words consisting of 14 LUT address bits and 2 control bits. This control scheme ***simplifies the hardware*** by allowing for the replacement of complex state machines, routed control signals, and supporting logic with special control data inserted directly into the data stream. Control data intermixed with application data is used to signal state transitions and trigger complex actions. LUT address bits select the appropriate SED. When building the query stream from a list of independent fragment formulae, the number of atoms of each unique element in a formula map to a separate word (e.g., H_2O would require two words while $C_3H_7NO_2S$ requires 5). Control bits distinguish between "begin," "middle," "end," and "done" states, triggering a different sequence of actions in each case (e.g., for $C_3H_7NO_2S$ encoding, the C_3 address with "begin" control form the 1st word, H_7NO_2 addresses and "middle" the next 3 words, while S and "end" the 5th word). Queries are positioned one after the other in the stream. When query stream encoding is complete, a dummy word with "done" control is inserted to signal hardware computations have completed and triggers a completion signal to the host.

Other design highlights. In an effort to eliminate BRAM as a limiting resource, only a single bank of LUTs is used to feed all Distribution Calculators (DCs) in a single FPGA device, where DC is the processing engine responsible for servicing a single query (specifics discussed in the next subsection). This approach is feasible because the LUT SED fetch rate far exceeds a DCs ability to process SEDs. Other than the initial round of query scheduling, DCs almost never stall waiting for SED input. Queries are scheduled to a selected DC by pushing SED and control data onto buffer FIFOs. DC selection is handled by a simple token-based round robin scheme, with in-stream detection of the "end" control state triggering token passage. Though configurable, FIFO depth need not be large to prevent DC starvation due to the aforementioned push/pop rate discrepancy.

B. Stage 2: Iterative combination for final distribution

Recall that the stage 1 output is equivalent to IPC output in the event that a chemical formula contains only a single element type. For molecules with multiple element types, individual peaks from the SEDs must be combined in a pairwise fashion to form the final distribution. Realizing the memory required to store full distributions for every probable query is unrealistic (e.g., the previous subset example would require a $256^4 \times 64 \times 16^{60}$ address space for full distributions vs. $4 \times 256 + 64 + 60 \times 16$ for SEDs), only SEDs are stored and combination to form the final distribution is necessary. With the LUT setup described in the previous subsection, as many clock cycles as elements in a query are required to retrieve all SED data needed for stage 2 calculation. Unfortunately, the resources required to handle SED combination of the worst-case query in the same number of clock cycles would be especially wasteful when handling the common-case, prompting us to move away from a fully-parallel, data-flow DC architecture towards an approach concentrating on iterative combination. This approach allows for a large number of DCs per FPGA operating at or near full hardware utilization rather than a small number operating with predominantly idle resources as with the alternative. Note that this approach also significantly increases the number of clock cycles a single DC requires to process a single query. Overall, in terms of total query throughput, iterative combination is far superior.

Figure 4 depicts our current DC approach. A general procedure for combination of two distributions, the accumulated final distribution thus far in the *Previous Iteration Buffer* and the current SED for combination as the next word in look-ahead FIFOs, following the Figure 4 approach is provided as Algorithm 3. *X* and *Y* are configuration parameters. *X* determines the number of parallel multipliers and adders instantiated in a DC. *Y* determines the maximum number of output peaks (refer to section III.C.). Increasing *X* reduces the total number of cycles required to process a single query at the expense of reduced clock rate by greatly complicating the *Insertion Sort*

Logic. Increasing *Y* increases the spectrum-window width while decreasing the number of DCs per FPGA because each buffer must grow to accommodate the larger number of possible output peaks. *X* and *Y* values should be chosen to maximize throughput while maintaining a large enough window to ensure scientific relevance.

The *Current Iteration Buffer* is designed as a bank of *Y* registers. *Previous iteration* and *Final Result* buffers are designed as parallel-loadable shift registers. *Result Reporting* circuitry operates independently so as not to stall calculation. New data loaded into the *Final Result Buffer* triggers the controller to shift result data into the *Collection FIFO* which must wait for the Result Serializer before inclusion into the output stream. As each output distribution can have a variable number of peaks, the controller searches passing peak values for the first zero probability as a break condition; the zero probability peak is also pushed onto the FIFO as a delimiter between adjacent distributions. If so desired, the aforementioned centroiding option is performed here, between Buffer and FIFO, assuming output is sorted by mass.

C. IPC top-level design and interaction with host

Shown in Figure 2 is the top-level design for the IPC. The *M* parameter in Figure 2, acting as a slack variable, should be chosen as to fill the target FPGA as much as possible without adversely affecting the achieved clock rate. As all design modules are separated by FIFOs, it is possible to place each in its own clock domain, in some cases improving performance.

In order to create an output stream of query distributions returned in the same order as query input, the token-based *Result Serializer* retrieves output from DCs in the same order as query scheduling, with in-stream detection of a zero probability peak triggering token passage. This aspect of the design allows compile-time parameter decisions to remain completely hidden from host interface code. Regardless of selected IPC core configuration, the host interface only sees a stream of delimited output distributions. Assuming the host interface builds the query stream correctly, a correct output stream will follow.

Most IPC software's perform computation with floating-point operands. Recognizing each peak has an associated probability bounded by 1.0 and a mass no larger than the longest combination of largest-mass SED peaks, we perform internal IPC calculation with variable precision fixed-point. The integer and fractional field lengths are configurable. Increasing mass integer bits increases the maximum representable mass. Increasing mass fractional bits decreases the minimum gap between adjacent peaks. Increasing probability fractional bits allows representation of lower probabilities. Reducing fractional bits reduces resource usage at the expense of result precision. An unintended benefit of reducing probability fractional bits is the imposition of an effective probability filter that removes peaks with low probabilities not representable with the chosen number of fractional bits. It is important to note that varying the bit widths for internal calculations does not affect LUT generation; LUT data is generated with the highest precision and only reduced inside the FPGA. This approach allows use of the same LUT initialization files with any hardware configuration.

The hardware is intentionally designed to operate independently of a specific SEDs identity in LUTs (i.e., no decision logic based on calculation of H_{233} vs. S_9 vs. others) allowing for complete separation of the query formula to LUT address mapping and hardware execution. This separation facilitates the ability to have different element subset partitions (realized as different LUT initialization files), yet use the same, unmodified hardware design. The only limitation on subset composition is that the total number of SEDs fit in a 16K address space (14-bits), considered more than sufficient for the current problem scope.

V. Experimental Evaluation on Novo-G

In this section, we evaluate the performance of our IPC solution. Although many de novo algorithms rely on IPC calculation, each one has slightly different expectations as to the provided functionality, and as such, a useful design must provide high customization and parameterization to accommodate as many algorithms and applications as possible. The previously discussed hardware architecture was implemented in VHDL and tested with various parameter configurations. Initial experiments were performed on a single Altera Stratix IV E530 FPGA on a GiDEL PROCStar IV board along with an Intel Xeon E5620 CPU for host support (results in Table I to be discussed). The single-device implementation was scaled and additional experiments were performed with a favorable IPC hardware configuration targeting multiple E530s, in multiple PROCStar IVs, in a single compute node of Novo-G, the reconfigurable supercomputer detailed in [18, 19] (results in Table II to be discussed). Finally, the implications of scaling to multiple compute nodes of Novo-G are discussed. The software baseline used for comparison is highly optimized, serial C++ code mirroring our hardware algorithm, executed on a single E5620 core. Originally, IPC software at [17] was our basis for comparison, but analysis of the fundamental IPC problem inspiring the architecture described in Section IV motivated us to develop software utilizing a similar algorithm. As expected, the developed software used for comparison has orders of magnitude better performance than [17] while maintaining the same accuracy. Output results from all hardware experiments were compared with those from the software baselines to confirm correctness.

A. Single-FPGA performance

Table I presents single-FPGA performance trends for various IPC parameter configurations. In the interest of brevity, only 7 of the most influential parameters are presented. 6 of the 7 variables are treated as independent. 1 of the 7, namely M, is used as a slack variable to fill the FPGA once all other variables have been selected. For consistency, all other parameters aside from the presented 7 are held constant, both in hardware and in the software baseline (e.g., a selected TP of 2^{-15} or sort-by-mass to enable centroiding). A set of 16×2^{20} generated fragment queries is used as input for comparison. The generation is not completely random, rather the relative abundance of each element type in amino acid structures is used to keep the ratios of H to C to N to O to S in each query statistically representative of actual amino acids.

Bandwidth limited configurations. Configurations 1-7 in Table I are generated without centroiding. In these configurations, the combined rate at which DCs generate peak data quickly exceeds the available bandwidth to external memory and host, and output serialization quickly becomes the bottleneck. The once computation-bound IPC problem in software becomes I/O-bound in FPGAs. For a given configuration, the minimum execution time has a lower bound fixed by the number of cycles required to store the total number of result peaks and increasing the peak production rate beyond this bound only increases DC idle time. This fact is most evident when comparing configurations 2 and 3 where, other than number of DCs and parallel computations per DC, configuration and clock rate are equal. As DCs in configuration 3 can process queries ~2× faster than configuration 2 (2× more parallel computations) one would expect the speedup/DC to be ~2× faster as well, resulting in an overall 1.5× greater speedup due to the 0.75× less DCs. Obviously this is not the case and the two configurations, operating at the same clock rate and producing the same results, execute in the same time. As operating with saturated output bandwidth stalls the DCs, skewing speedup/DC, the corresponding column for configurations 1-7 is intentionally unreported.

This saturated bandwidth issue is an obvious limitation when integrating with de novo methods that require full distribution output. We plan to address this issue in the future by integrating our IPC solution directly with a theoretical spectrum generator, allowing us to bring more of the algorithm on-chip, removing the need to serially stream all results back to the host. In order to experimentally measure current speedup/DC accurately, we instantiate a centroiding module on the backend of each DC which does not alter the total number of DC calculations but does reduce the number of peaks in the final distribution by an order of magnitude. Experiments for configurations 8-20 are performed with centroiding.

Calculation word width. Configurations 8-10 demonstrate the degree to which reducing word width for internal calculations reduces logic and increases achievable clock rate. The reduction in logic is almost linear as expected but the increase in clock frequency displays a trend of diminishing returns. The bit reduction from configuration 8 to 9 (32+32 total bits to 26+24) allows for a sizable increase in frequency whereas moving lower, as in configuration 9 to 10 (26+24 total bits to 22+16), provides only a minor benefit, plus the disadvantage of lower result precision.

Parallel computations per DC. 11-13 demonstrate the effect of increased parallel operations per cycle (X). As expected, the total number of cycles required to compute a distribution is reduced (i.e., increased speedup/DC). However, increasing the number of parallel operations too high ($X \geq 3$) negatively impacts logic usage, routability, and design frequency not only because of the increased number of multipliers and adders required to calculate the additional peaks in parallel, but also because each new peak must then be sorted and inserted into the result buffer in parallel. This leads to a decrease in DCs per FPGA operating at a slower rate. Increasing parallel computations beyond two is seen to degrade overall performance rather than improve it.

TABLE I. Single-FPGA performance for several parameter configurations.

	Configuration*							Freq (MHz)	Speedup† /DC	Speedup† /FPGA
	N	X	Y	Qi.f$_{Mas}$	Qi.f$_{Prob}$	Cen	M			
1.	16	1	128	16.16	1.31	N	12	115	–	72
2.	16	1	128	14.8	1.15	N	20	145	–	115
3.	16	2	128	14.8	1.15	N	15	145	–	115
4.	16	3	128	14.8	1.15	N	10	110	–	87
5.	12	2	128	14.8	1.15	N	14	155	–	123
6.	16	2	80	14.8	1.15	N	21	150	–	120
7.	12	2	80	14.8	1.15	N	22	155	–	127
8.	16	1	128	16.16	1.31	Y	11	120	17	186
9.	16	1	128	14.12	1.23	Y	13	135	18	236
10.	16	1	128	14.8	1.15	Y	16	140	24	384
11.	16	1	128	14.12	1.23	Y	13	135	18	236
12.	16	2	128	14.12	1.23	Y	10	130	32	325
13.	16	3	128	14.12	1.23	Y	6	100	35	214
14.	16	2	128	14.12	1.23	Y	10	130	32	325
15.	12	2	128	14.12	1.23	Y	10	135	34	338
16.	8	2	128	14.12	1.23	Y	11	135	35	381
17.	16	2	100	14.12	1.23	Y	13	135	34	444
18.	16	2	80	14.12	1.23	Y	14	130	32	454
19.	16	2	60	14.12	1.23	Y	17	130	33	566
20.	12	2	80	14.12	1.23	Y	15	135	34	516

* N: max peaks/SED, X: number of parallel peak computations per DC, Y: max peaks/output-window, Qi.f: fixed-point word width bits (integer.fractional), Cen: centroiding capability enabled (Yes/No), M: DCs per FPGA.

† w.r.t. C++ software processing $16x2^{20}$ statistically representative queries (based on relative elemental abundance in amino acids) in 821 seconds on a single E5620 core.

Distribution window width. 14-19 demonstrate the effect of reducing SED input (N) and/or final distribution output (Y) window sizes. Although no real effect on design frequency is observed, there is linear savings on logic usage per DC at the expense of result exactness. Speedup/DC remains steady throughout these experiments, but as window size is decreased, it is possible to fit more DCs per FPGA, increasing overall performance. Reducing window size too low increases the possibility that useful peak data may be trimmed from the final output as opposed to the inessential peak data possibly trimmed when window size is higher. A tradeoff between performance and scientific relevance must be considered. Configuration 20 represents a suitable *sweet spot*, achieving remarkable speedup (516) while ensuring results remain scientifically relevant.

B. Multi-FPGA performance

If an encountered problem requires even faster execution than possible with a single device, multiple FPGAs may be utilized. Each query computation is completely independent from one to the next and as such, the described design should be completely scalable given a sufficiently large dataset. As the design produces large amounts of data that currently must be transferred back to the host, limited system resources and I/O rates can potentially constrain the designs full scalability.

Table II lists multi-FPGA performance of the scaled design for several system configurations on a single node of

TABLE II. Multi-FPGA performance for several node configurations.

	Boards /Node	FPGAs /Board	Total FPGAs	Speedup† /FPGA	Total Speedup†
1.	1	1	1	517	517
2.	1	2	2	516	1031
3.	1	3	3	398	1192
4.	1	4	4	315	1259
5.	2	1	2	516	1033
6.	2	2	4	502	2009
7.	2	4	8	313	2510
8.	4	1	4	492	1968
9.	4	4	16	209	3340

† w.r.t. C++ software processing 2^{30} statistically representative queries (based on relative elemental abundance) in 52,478 seconds on a single E5620 core.

Novo-G (i.e., up to 16 FPGAs on 4 PROCStar IV boards). Using 2 threads (producer/consumer model) per enabled FPGA for single-device control, we demonstrate multi-FPGA performance by processing a large data set (2^{30} queries producing nearly 62 GB of output), with configuration 20 from Table I, on varying quantities of resources.

Scaling FPGAs per board limited by I/O bandwidth. When scaling the design to multiple FPGAs on a single PROCStar IV board (rows 1-4), the achieved speedup is linearly proportional to the number of FPGAs assuming I/O bandwidth is not saturated (rows 1 & 2). However, as the number of FPGAs increases beyond the point where the amount of output data produced per unit time exceeds the I/O bandwidth available to a single board, additional time beyond that which is needed to perform the actual IPC calculations is needed to stream output from FPGA to host. The reduction in multi-FPGA performance caused by limited I/O bandwidth is evident when comparing the linear performance trend of rows 1 and 2 with the sub-linear trend of rows 2−4. The effect is especially pronounced when scaling an already bandwidth limited single-FPGA configuration such as 7 from Table I to 4 FPGAs; in this case only a speedup of 173 vs. software (not shown in Table II) is achieved when processing the Table II queries as opposed to the best-case single-board speedup of 1259 reported in row 4. In these experiments, the I/O bottleneck is identified as the board link to the system bus. As the PROCStar IV only supports 8 lane, Gen 1 PCIe, we would expect more scalability with a system configuration employing more lanes and/or the more recent Gen 3 PCIe standard.

Scaling boards per node limited by CPU resources. Because the available system bandwidth exceeds that of the board link bottleneck, the single-board trends remain while the multi-board performance scales linearly when employing multiple boards per node (rows 5−9). There is a slight deviation from the expected trend in row 9, now caused by limited CPU resources rather than limited bandwidth. The row 9 experiment requires 32 total threads (2 per FPGA), 16 of which (consumers) require most of the cycles, executing in parallel on 2 quad-core E5620s with only a total of 8 physical cores (16 logical due to hyper-threading). Note that by limiting overhead associated with context switching and memory thrashing, we would expect more scalability with a system configuration employing more physical cores. By employing a single Novo-G node, we increase the single-FPGA speedup of 517 to a remarkable 3340.

Multi-node scaling. Because there is no communication between nodes required by the IPC algorithm assuming input queries are already pre-partitioned, the only multi-node overhead incurred when scaling to multiple servers would be the time required to transmit an initialization signal to each node and the time required to synchronize when complete. Previous single-board and single-node trends should continue while performance scales linearly with additional nodes. By employing multiple Novo-G nodes, we would expect best-case performance to increase beyond 3340 almost linearly with minor overhead and without accompanying performance limiters such as I/O bandwidth (FPGA/board scaling) and CPU resources (board/node scaling). We plan to verify our expectations by scaling our design to multiple compute nodes in Novo-G as future work.

Multi-FPGA advantage. One final advantage of multi-FPGA execution (not demonstrated in Table II) over a single FPGA is the ability to load each FPGA with a different bit file and execute different configurations in parallel. Although a batch of IPC queries generally require the same precision, threshold probability, etc., when used in the same higher order identification algorithm, only the largest of queries require a large distribution window (e.g., Table I configuration 12) while the more common, smaller queries require only a moderately sized one (e.g., configuration 20). As reducing window width allows more DCs per FPGA and consequently more speedup/FPGA, scheduling queries to those FPGAs with the most appropriate configurations rather than all queries to a single configuration selected to handle the worst case will have the effect of increasing overall query throughput.

VI. SUMMARY, CONCLUSIONS, AND FUTURE WORK

In this paper, we presented the first FPGA-based Isotope Pattern Calculator, a computationally intense subroutine common in de novo protein identification. With a VHDL implementation of the presented design, we demonstrated between 72 and 566 speedup on a single FPGA, up to 1259 speedup on a single board (4 FPGAs), and up to 3340 on a single node (16 FPGAs), when compared to a highly optimized, serial C++ IPC implementation. The wide range of achieved single-FPGA performance is due to the varying of several key configuration parameters that allow the design to adapt to the needs of multiple protein identification algorithms. The wide range of achieved single-node performance is due to embarrassingly parallel algorithm scalability restricted by real-world system limitations such as insufficient I/O bandwidth and CPU resources. Lessening the encountered system limitations should increase performance.

In general, most current de novo algorithms would not require more than one FPGA to one board (i.e., 4 FPGAs) to reduce execution time of the IPC subroutine for a single protein identification down to something reasonable, but just imagine what new algorithms this level of performance can enable. Previously dismissed approaches with potentially revolutionary accuracy yet obscene execution time now reach the level of current in-use algorithms with the aid of a few FPGAs. With a system comparable to Novo-G (with

nearly 400 FPGAs in 2012), these algorithms now become practical (assuming the potential improvement in accuracy is realized). Furthermore, envision a scenario in which a large proteomics lab offering protein identification as a service for personalized medicine or cancer diagnostics must service large volumes of queries. Rather than run and maintain a massively large, energy-hungry, and expensive HPC cluster consisting exclusively of conventional x86 devices, deploying a system consisting of both x86 and high-performance FPGA devices would allow adequate performance to handle the large volume sustainably, facilitating a considerably lower total cost of ownership. However, there is still much work to be done before this scenario is a reality for protein identification.

The work described offers many possibilities for future study. Clearly, the next logical step is to integrate the IPC core into a de novo identification algorithm, accomplished by first integrating with a full theoretical spectrum generator. This would involve moving more of the algorithm onto FPGAs. Another direction under consideration is implementation and testing of an already conceived preliminary design for a GPU accelerated IPC. With a few minor modifications to the base algorithm in Section III, we believe IPC is at least moderately GPU amenable and hopefully testing of our preliminary design will justify further study. As sorting is fundamentally integrated into the current DC design, a non-sorted output option was never attempted. Designing a DC component that does not rely on sorting would allow for much greater parallelization with far less resources, resulting in greater speedup for those higher order algorithms utilizing IPC without the need for sorted output. Finally, the previously omitted task of scaling our current FPGA design to multiple Novo-G nodes will be performed.

ACKNOWLEDGMENT

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. EEC-0642422. We thank Jason Hachen for his considerable efforts in optimizing the IPC software baseline. We thank Kent Koeninger and Martin Lewitt at Veritomyx for their support, detailed problem specification, and guidance during the design process to ensure scientific relevance.

REFERENCES

[1] W. P. Blackstock and M. P. Weir, "Proteomics: Quantitative and Physical Mapping of Cellular Proteins," Trends in Biotechnology, vol. 17, issue 3, pg. 121-127, 1999.

[2] F. Forner, L. J. Foster, and S. Toppo, "Mass Spectrometry Data Analysis in the Proteomics Era," Current Bioinformatics, vol. 2, pp. 63-93, 2007.

[3] L. Lancashire, O. Schmid, H. Shah, and G. Ball, "Classification of bacterial species from proteomic data using combinatorial approaches incorporating artificial neural networks, cluster analysis and principal components analysis," Bioinformatics, vol. 21, issue 10, pp. 2191-2199, 2005.

[4] A. W. Bell *et al.*, "A HUPO test sample study reveals common problems in mass spectrometry-based proteomics," Nat. Methods, vol 6, pp. 423-430, 2009.

[5] E. A. Kapp *et al.*, "An evaluation, comparison, and accurate benchmarking of several publicly available MS/MS search algorithms: Sensitivity and specificity analysis," Proteomics, vol 5, pp. 3475–3490, 2005.

[6] D. N. Perkins, D. J. Pappin, D. M. Creasy, and J. S. Cottrell, "Probability-based protein identification by searching sequence databases using mass spectrometry data," Electrophoresis, vol. 20, pp. 3551-3567, 1999.

[7] D. Lopez-Ferrer, S. Martinez-Bartolome, M. Villar, M. Campillos, F. Martin-Maroto, and J. Vazquez, "Statistical model for large-scale pep- tide identification in databases from tandem mass spectra using SEQUEST," Anal Chem, vol. 76, pp. 6853-6860, 2004.

[8] B. Ma *et al.*, "PEAKS: powerful software for peptide de novo sequencing by tandem mass spectrometry," Rapid Communications in Mass Spec., vol. 17, pp. 2337-2342, 2003.

[9] P. Hernandez, M. Muller, and R. D. Appel, "Automated Protein Identification by Tandem Mass Spectrometry: Issues and Strategies," Wiley InterScience, 2005.

[10] M. L. Brownawell and J. S. Filippo, "A Program for the Synthesis of Mass-Spectral Isotopic Abundances," J. Chem. Educ., vol. 59, pp. 663-665, 1982.

[11] R. K. Snider, "Efficient Calculation of Exact Mass Isotopic Distributions," J. of the Amer. Soc. for Mass Spec., vol. 18, issue 8, pp. 1511-1515, Aug 2007.

[12] A. L. Rockwood, S. L. Van Orden, and R. D. Smith, "Ultrahigh Resolution Isotope Distribution Calculations," Rapid Communications in Mass Spec., vol. 10, pp. 54 –59, 1996.

[13] A. L. Rockwood, S. L. Van Orden, and R. D. Smith, "Rapid Calculation of Isotope Distributions," Anal. Chem., vol. 67, pp. 2699–2704, 1995.

[14] A. L. Rockwood, "Relationship of Fourier-Transforms to Isotope Distribution Calculations," Rapid Communications in Mass Spec., vol. 9, pp. 103–105, 1995.

[15] J. Yergey, "A General Approach to Calculating Isotopic Distributions for Mass Spectrometry," Int. J. Mass Spec. Ion Physics, vol. 52, pp. 337-349, 1983.

[16] H. Kubinyi, "Calculation of Isotope Distributions in Mass Spectrometry. A Trivial Solution for a Nontrivial Problem," Anal. Chim. Acta., vol. 247, pp. 107–119, 1991.

[17] Dirk (2005), *Isotopic Pattern Calculator,* http://isotopatcalc.sourceforge.net/index.php, File: gips-0.7.tar.gz.

[18] A. George, H. Lam, A. Lawande, C. Pascoe, and G. Stitt, "Novo-G: A View at the HPC Crossroads for Scientific Computing," Proc. of the Int. Conf. on Eng. of Reconf. Sys. and Algs. (ERSA), NV, 2010.

[19] A. George, H. Lam, and G. Stitt, "Novo-G: At the Forefront of Scalable Reconfigurable Computing," IEEE Computing in Sci. & Eng. (CiSE), Vol. 13, No. 1, Jan/Feb. 2011, pp. 82-86.

[20] L. McHugh and J. W. Arthur, "Computational Methods for Protein Identification from Mass Spectrometry Data," PLoS Comp. Biology, 2008.

[21] J. N. Louris, L. G. Wright, and R. G. Cooks, "New Scan Modes Accessed with a Hybrid Mass Spectrometer," Anal. Chem., vol. 57, pp. 2918-2924, 1985.

[22] A. M. Falick, W. M. Hines, K. F. Medzihradszky, M. A. Baldwin, and B. W. Gibson, "Low-Mass Ions Produced from Peptides by High-Energy Collision-Induced Dissociation in Tandem Mass Spectrometry," J. of the Amer. Soc. for Mass Spec., vol. 4, issue 11, pp. 882-893, 1993.

[23] Y. Shimonishi *et al.*, "Sequencing of peptide mixtures by Edman degradation and field-desorption mass spectrometry," Eur. J. Biochem., 112(2): 251–264, 1980.

[24] D. L. Tabb, C. Narasimhan, M. B. Strader, and R. L. Hettich, "DBDigger: reorganized proteomic database identification that improves flexibility and speed," Anal. Chem., vol. 77, pp. 2464-2474, 2005.

[25] V. Dancik, T. A. Addona, K. R. Clauser, J. E. Vath, and P. E. Pevzner, "De novo peptide sequencing via tandem mass spectrometry," J. Comput. Biol., vol. 6, pp. 327-342, 1995.

[26] A. Frank and P. Pevzner, "PepNovo: de novo peptide sequencing via probabilistic network modeling," Anal. Chem., vol. 77, pp. 964-973, 2005.

[27] M. Mann and M. Wilm, "Error-tolerant identification of peptides in sequence databases by peptide sequence tags," Anal. Chem., vol. 66 pp. 4390–4399, 1994.

[28] A. Frank, M. Savitski, M. Nielsen, R. Zubarev, and P. Pelzner, "De Novo Peptide Sequencing and Identification with Precision Mass Spectrometry," Proteome Research, vol. 6, pp. 114-123, 2006.

[29] C. Pascoe *et al.*, "Reconfigurable supercomputing with scalable systolic arrays and in-stream control for wavefront genomics processing," Proc. of Symposium on Application Accelerators in High-Performance Computing (SAAHPC), TN, 2010.