آشکارسازی سیگنال بر اساس پردازش موازی مبتنی بر جی‌پی‌یو در شبکه‌های حس‌گری صوتی دارای زیرساخت

حامد صادقی[1*] و امیر اخوان[2]

[1] آزمایشگاه مخابرات بی‌سیم، دانشکده برق و کامپیوتر، دانشگاه تربیت مدرس، تهران، ایران

[2] گروه مهندسی پزشکی، دانشگاه صنعتی همدان، همدان، ایران

چکیده

الگوریتم فیشر، یکی از معروف‌ترین و پرکاربردترین روش‌های آشکارسازی آرایه‌ای سیگنال‌های صوتی بسامدِ پایین در شبکه‌های حس‌گری دارای زیرساخت است؛ اما یکی از مشکلات عمده در به‌کارگیری این الگوریتم، زمان طولانی انجام پردازش در آن است که درعمل، پیاده‌سازی بلادرنگ آشکارساز را با مشکل مواجه می‌سازد. در این مقاله، چگونگی پیاده‌سازی الگوریتم فیشر را با استفاده از واحد پردازش گرافیک (جی‌پی‌یو) به‌منظور تحقق محاسبات سریع و انجام پردازش‌های نزدیک به زمانِ واقعی، ارائه می‌کنیم. به‌خصوص به‌منظور بهبود هرچه بیشتر سرعت محاسبات، الگوریتم آشکارسازی با استفاده از روش پردازش موازی (مبتنی بر جی‌پی‌یو) پیاده‌سازی شده است. نتایج شبیه‌سازی‌ها، ارتقای قابل ملاحظه سرعت آشکارساز فیشر را نشان می‌دهند که باعث بهبود کارآیی شبکه حس‌گری صوتی خواهد شد.

واژگان کلیدی: شبکه حس‌گری، پردازش آرایه‌ای، شکل‌دهی پرتو، پردازش موازی، جی‌پی‌یو.

Signal Detection Based on GPU-Assisted Parallel Processing for Infrastructure-based Acoustical Sensor Networks

Hamed Sadeghi[1*] & Amir Akhavan[2]

[1] Wireless Communication Laboratory, ECE Department, Tarbiat Modares University, Tehran, Iran

[2] Department of Biomedical Engineering, Hamedan University of Technology, Hamedan, Iran

Abstract

Nowadays, several infrastructure-based low-frequency acoustical sensor networks are employed in different applications to monitor the activity of diverse natural and man-made phenomena, such as avalanches, earthquakes, volcanic eruptions, severe storms, super-sonic aircraft flights, etc. Two signal detection methods are usually implemented in these networks for the purpose of event occurrence identification, which are the progressive multi-channel correlator (PMCC) and the so-called Fisher detector. But, the Fisher method is more important and applicable in low signal-to-noise (SNR) ratio conditions, which is of a special interest in acoustical monitoring networks. Unfortunately, an important disadvantage of this algorithm is its relative high detection-time; which limits its application for real-time detection scenarios. This disadvantage is fundamentally due to a beam forming process in Fisher algorithm, which requires doing complete search in a slowness-network, constructed from possible incoming wave front directions and speeds. To address this issue, we propose a method for implementation of this beam forming on a graphics processing unit (GPU), in order

* نویسندهٔ عهده‌دار مکاتبات

* Corresponding author

to realize a fast-computing and/or near real-time signal processing technique. In addition, we also propose a parallel-processing algorithm for further enhancement of the performance of this GPU-based Fisher detector. Simulation results confirm the performance improvement of Fisher detector, in terms of required processing time for acoustical signal detection applications.

Keywords: **Sensor network, array processing, beamforming, parallel processing, GPU.**

۱- مقدمه

یکی از انواع شبکه‌های حس‌گری[1]، شبکه‌های حس‌گری صوتی مبتنی بر زیرساخت[2] (IASN) است که بیش‌تر به‌منظور آشکارسازی و جهت‌یابی سیگنال‌های گسیل‌شده از منبع مولد صدا به‌کار می‌روند [1]- [4]. در IASN پس از دریافت سیگنال‌های ارسالی از حس‌گرها در مرکز هم‌جوشی[3] (FC)، وقوع رخداد با استفاده از الگوریتم‌های پردازش آرایه‌ای جستجو می‌شود. این الگوریتم‌ها شامل دو مرحله اصلی آشکاری‌سازی[4] و سپس تخمین جهت و سرعت ورود موج است. اصول پردازش سیگنال در این الگوریتم‌ها بر مبنای استفاده از مفهوم هم‌دوسی[5] امواج ساطع‌شده از منبع صوت و نیز هم‌بستگی متقابل بین سیگنال‌های خروجی حس‌گرها است. به‌عبارت دیگر، از آن‌جایی‌که در صورت حضورسیگنال مورد نظر در موج‌دیس‌های ثبت‌شده، هم‌بستگی بین موج‌دیس‌های خروجی حس‌گرها افزایش می‌یابد، لذا پارامتر هم‌بستگی به‌عنوان یک معیار مناسب برای جداکردن نوفه از سیگنال هدف، می‌تواند استفاده شود. از طرفی به‌منظور تخمین اختلاف زمان ورود سیگنال به حس‌گرهای متفاوت از هم‌بستگی متقابل بین سیگنال‌های حس‌گرهای مختلف می‌توان استفاده کرد. روشن است که مقادیر اختلاف زمانی، به هندسه و فاصله نسبی حس‌گرها از یکدیگر و نیز پارامترهای موج صوتی وابسته است. پس از تخمین مقادیر اختلاف زمانی، پارامترهای انتشار موج از جمله راستای ورود موج و سرعت آن محاسبه می‌شود.

یکی از روش‌های پرکاربرد آشکارسازی سیگنال و تخمین پارامترهای موج در IASN، الگوریتم فیشر[6] است. این روش به‌دلیل پایداری مناسب در مقابل نوفه و قابلیت پیاده‌سازی پهن‌باند به‌طور گسترده برای تعیین جهت و آشکارسازی سیگنال‌های صوتی و لرزه‌ای به‌کار می‌رود [5]- [8]. روش فیشر برای نخستین‌بار در سال ۱۹۵۷ توسط *ملتن* و *بایلی*[7] پیشنهاد شد [9]. پس از آن این الگوریتم به‌منظور آشکارسازی سیگنال‌های لرزه‌ای استفاده شد [10]. این آشکارساز که بر اساس آنالیز واریانس نسبت سیگنال به نوفه[8] استخراج می‌شود، شامل دو بخش شکل‌دهی پرتو[9] و آشکارسازی مبتنی بر نسبت فیشر (به‌منظور ارزیابی محتوای سیگنال پرتوها) است. نسبت فیشر درواقع نسبت واریانس سیگنال به واریانس نوفه است که در هر دو حوزه زمان و بسآمد قابل پیاده‌سازی است. یکی از ویژگی‌های برجسته این روش آشکارسازی، مقاوم‌بودن آن در مقابل نوفه است. این خاصیت در پردازش داده‌های مربوط به زلزله و شرایطی که نسبت توان سیگنال به نوفه پایین است، اهمیت زیادی دارد. شایان ذکر است که آشکارساز فیشر در کاربردهای مختلفی هم‌چون تشخیص فعالیت‌های آتشفشانی و ردیابی مسیر شهاب‌های آسمانی نیز به‌کار می‌رود [11]- [12]. این روش در مقایسه با آشکارساز معروف و پرکاربرد هم‌بستگی چندکاناله پیشرو[10]، موسوم به پی‌ام‌سی‌سی [13]، توانایی بیش‌تری برای آشکارسازی سیگنال در نسبت سیگنال به نوفه‌های پایین دارد. در مقابل، الگوریتم فیشر به‌دلیل جستجوی صفحه کُندی به زمان بیشتری برای پردازش داده نیاز دارد.

در روش آشکارساز فیشر تخمین پارامترهای موج دریافتی بر مبنای اختلاف زمانی بین ثبت‌های صورت‌گرفته در حس‌گرها انجام می‌شود. از آن‌جایی که این اختلاف زمانی، به چیدمان حس‌گرهای آرایه و جهت ورود سیگنال بستگی دارند، در این روش ثبت‌های حس‌گرها به‌ازای یک

[1] Sensor Networks (SNs)

[2] Infrastructure-based Acoustic Sensor Network (IASN)

[3] Fusion Center (FC)

[4] Detection

[5] Coherency

[6] Fisher

[7] Melton and Bailey

[8] Signal-to-Noise Ratio (SNR)

[9] Beamforming

[10] Progressive Multi-Channel Correlation (PMCC)

جهت و سرعت فرضی برای ورود سیگنال، هم‌تراز می‌شوند؛ سپس هم‌دوسی آن‌ها با استفاده از نسبت فیشر محاسبه می‌شود. این کار به‌طور مداوم به‌ازای جهت‌ها و سرعت‌های مختلف ورود سیگنال صدا صورت می‌پذیرد تا درنهایت، جهت و سرعتی که بیشینه مقدار هم‌دوسی بین ثبت‌ها را ایجاد کند، به‌عنوان جهت و سرعت موج انتخاب شود. یک روش جستجوی سامان‌مند[1] برای تخمین جهت و سرعت موج، استفاده از روش شکل‌دهی پرتو است که در آن از یک شبکه کُندی[2] استفاده می‌شود. اندازه هر بردار در صفحه کُندی، معکوس سرعت ظاهری[3] است و زاویه آن با شمال زاویه سمت‌پشتی[4] نامیده می‌شود. جستجوی بردار کندی در تمامی جهت‌ها، تضمین می‌کند که یکی از پرتوها جهت تقریبی ورود سیگنال به آرایه را نشان دهد. واضح است که بسته به تعداد نمونه‌ها در صفحه کندی، میزان تفکیک‌پذیری[5] زاویه و سرعت تخمینی موج در این شبکه تحت تاثیر قرار خواهد گرفت. از سوی دیگر با افزایش تعداد نمونه‌ها، زمان بیشتری نیز برای محاسبات مورد نیاز است و این مانع از پیاده‌سازی بلادرنگ آشکارسازی سیگنال‌های صوتی می‌شود.

ازآنجایی‌که مشکل اصلی روش فیشر، انجام جستجوی سنگین[6] در فضای شبکه کُندی و درنتیجه، غیر زمانِ واقعی‌بودن سامانه است، در این مقاله یک روش سریع‌سازی الگوریتم فیشر پیشنهاد می‌شود که دارای کاربردهای گسترده‌ای در سامانه‌های نظارتی خواهد بود. روش کار بدین‌صورت است که ابتدا برخی محاسبات اولیه و نیز پنجره‌گذاری بر روی داده‌های ورودی در سی‌پی‌یو انجام شده و سپس برای هر پنجره، نسبت‌های آمارگان فیشر به‌ازای هر کدام از عناصر شبکه کُندی، به‌طور موازی توسط جی‌پی‌یو محاسبه شوند. درنهایت، بردار کُندیِ متناظر با بیشینه نسبت فیشر، انتخاب شده و با استفاده از آن پارامترهای موج برای آن پنجره زمانی تخمین زده می‌شوند. در این مقاله نشان داده خواهد شد که روش پیشنهادی باعث کاهش زمان آشکارسازی منبع شد، که این امر برای سامانه‌های نظارتی بسیار حائز اهمیت است. به‌عنوان جمع‌بندی، این روش یک الگوریتم تلفیقی شکل‌دهی پرتوی سریع مبتنی بر آمارگان فیشر خواهد بود که به‌طور همزمان، هم حضور سیگنال هدف را آشکارسازی کرده و هم آن‌را می‌تواند جهت‌یابی کند.

ساختار مقاله بدین شرح است: در بخش ۲، مفاهیم اولیه پردازش موازی داده، ساختار پردازنده گرافیکی و شیوه برنامه‌نویسی آن شرح داده شده است. همچنین در این بخش مروری مختصر بر آشکارسازی سیگنال‌های بسآمد-پایین با استفاده از نسبت فیشر انجام می‌شود. در بخش ۳ روش پیشنهادی آشکارسازی سریع منابع صوتی بسآمدِ پایین و چگونگی پیاده‌سازی الگوریتم آن ارائه شده است. نتایج شبیه‌سازی‌ها و کارآیی روش پیشنهادی در بخش ۴ بررسی می‌شوند. در انتها نتیجه‌گیری کلی مقاله و مراجع در بخش ۵ و ۶ ارائه شده است.

۲- آشکارسازی سریع سیگنال‌های صوتی

درهمین‌اواخر استفاده از واحد پردازنده گرافیکی (جی‌پی‌یو[7]) مورد توجه پژوهش‌گران حوزهٔ محاسبات و پردازش سریع قرار گرفته است. مقالات منتشرشده در این زمینه حاکی از آن است که به‌کارگیری این پردازنده‌ها به‌منظور تسریع الگوریتم‌های دارای قابلیت موازی‌سازی، موفق بوده است [14]- [19]. به‌طور خاص، در حال حاضر یکی از زمینه‌های پژوهشی فعال در حوزه‌هایی که نیازمند پردازش سریع هستند و قابلیت موازی شدن در الگوریتم‌های مورد استفاده آن‌ها وجود دارد، به‌کارگیری پردازنده‌های گرافیکی با معماری کودا[8] است [20]،[21]. در این مقاله پیشنهاد می‌شود، از این پردازنده‌ها به‌منظور پردازش سریع سیگنال‌های صوتی دریافتی از IASNها استفاده شود. به‌عبارت دیگر در این مقاله، الگوریتم فیشر با استفاده از پردازش موازی مبتنی بر جی‌پی‌یو پیاده‌سازی شده است. نتایج مقاله نشان می‌دهد که با این روی‌کرد، نقطه ضعف آشکارساز فیشر، که زمان‌بربودن جستجو در شبکه کُندی است، برطرف شده است.

پیش از پرداختن به الگوریتم پیشنهادی، ابتدا برخی مفاهیم پایه‌ای معرفی می‌شوند.

[1] Systematic
[2] Slowness
[3] Apparent velocity
[4] Back azimuth (BAZ)
[5] Resolution
[6] Exhaustive search
[7] Graphics Processing Unit (GPU)
[8] Compute Unified Device Architecture (CUDA)

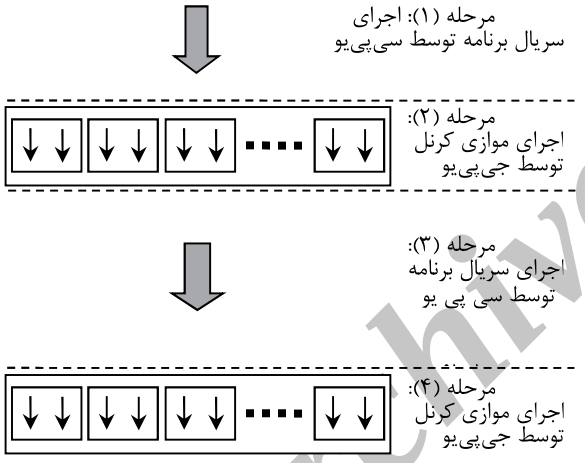
۱-۲- موازی‌سازی داده‌ها

در بسیاری از کاربردها حجم بالای داده، زمان پردازش نرم‌افزارهای موجود را در رایانه‌های سری بسیار طولانی می‌کند. هم‌چنین بسیاری از این نرم‌افزارها پدیده‌های فیزیکی موجود در دنیای واقعی را مدل می‌کنند. به‌عنوان نمونه، تصاویر و فریم‌های ویدئویی، نمایش لحظه‌ای دنیای واقعی هستند که در آن بخش‌های مختلف تصویر، پدیده‌های فیزیکی مستقل را به‌طور هم‌زمان نشان می‌دهند. چنین استقلالی، پایه‌ی موازی‌سازی داده‌ها است. یک مثال ساده برای درک موازی‌سازی داده‌ها، ضرب دو ماتریس بسیار بزرگ در یکدیگر است که در آن هر درایه ماتریس حاصل ضرب، به‌صورت مستقل از سایر درایه‌ها قابل محاسبه است. به عبارت دیگر عملیات ریاضیِ لازم را برای محاسبه تمام درایه‌های ماتریس خروجی به‌صورت موازی می‌توان انجام داد؛ بنابراین ضرب دو ماتریس با ابعاد بالا قابلیت زیادی در موازی‌سازی داده دارد. موازی‌سازی داده در کاربردهای واقعی می‌تواند بسیار پیچیده‌تر باشد.

۲-۲- واحد پردازنده گرافیکی با معماری کودا

جی‌پی‌یو یک پردازنده موازی است که برای کاربردهای گرافیکی ابداع شده است؛ اما در سال‌های اخیر به‌عنوان یک پردازنده عمومی نیز به کار گرفته می‌شود. تفاوت عمده و ساختاری بین جی‌پی‌یو و سی‌پی‌یو[1] در این است که معماری سخت‌افزاری جی‌پی‌یو تعداد بیش‌تری آ‌ل‌یو[2] را دربردارد و در مقابل، اجزای کم‌تری برای کَش[3] و واحد کنترل دارد. این امر باعث می‌شود تا جی‌پی‌یو قابلیت محاسباتی بالایی داشته باشد. به‌طور خاص، جی‌پی‌یو توان‌مندی بیش‌تری برای اجرای موازی محاسبات ریاضی، که در آن تعداد زیادی عملیات مشابه، روی تعداد زیادی داده انجام می‌شود، دارد [22].

همان‌طور که بیان شد، نسل‌های اولیه جی‌پی‌یو تنها قادر به اجرای محاسبات گرافیکی بودند؛ امّا از اواخر سال ۲۰۰۷، معماری کودا توسط شرکت اِن‌ویدیا[4] به‌منظور بهره‌گیری از قابلیت‌های محاسباتی جی‌پی‌یو در سایر کاربردها، ابداع شد. با استفاده از این معماری، جی‌پی‌یو می‌تواند علاوه بر محاسبات گرافیگی، محاسبات همه‌منظوره را نیز انجام دهد. شرکت اِن‌ویدیا علاوه بر ارائه سخت‌افزاری که بتواند محاسبات گرافیکی و همه‌منظوره را انجام دهد، زبان برنامه‌نویسی جدیدی به نام کودا- سی[5] را نیز ارائه کرد. این نوع برنامه‌نویسی که در قالب محاسباتی غیرهمگن[6] انجام می‌شود، امکان بهره‌گیری هم‌زمان از مزایای سی‌پی‌یو و جی‌پی‌یو را فراهم می‌سازد. در این روشِ برنامه‌نویسی، بخش سری کد توسط سی‌پی‌یو اجرا شده و هسته موازی برنامه، موسوم به کِرنل، توسط جی‌پی‌یو پردازش می‌شود. مدل اجرایی این روش برنامه‌نویسی در (شکل-۱(۱) نمایش داده شده است. با توجه به این شکل، سی‌پی‌یو کدهای مربوط به مراحل ۱و ۳ را به‌صورت سریال و جی‌پی‌یو کدهای مربوط به مراحل ۲ و ۴ را به صورت موازی اجرا می‌کند. هنگامی که یک تابع کِرنل به‌صورت موازی راه‌اندازی[7] شود (مرحله ۲ و ۴)، تعداد زیادی thread بر روی جی‌پی‌یو اجرا می‌شود.



(شکل-۱): مدل برنامه‌نویسی کودا- سی.
(Figure-1): CUDA-C programming model.

مجموعه تمام threadهایی که توسط یک تابع کِرنل ایجاد می‌شود، گرید[8] نامیده می‌شود. هنگامی که اجرای تمام threadهای یک کِرنل پایان یابد، اجرای برنامه بر روی میزبان (سی‌پی‌یو) آغاز شده و تا زمان اجرای کِرنل بعدی ادامه می‌یابد.

[1] Central Processing Unit (CPU)
[2] Arithmetic Logic Unit (ALU)
[3] Cache
[4] NVIDIA
[5] CUDA C
[6] Heterogeneous computing
[7] Launch
[8] Grid

۳-۲- آماره آشکارساز فیشر

در پردازش بلادرنگ سیگنال‌های صوتی، نه تنها آشکارسازی باید به اندازهٔ کافی سریع باشد، بلکه باید روی‌داد به صورت خودکار و بدون نیاز به مفسر تشخیص داده شود. یک راه برای انجام این کار استفاده از آشکارساز آماری فیشر است که به‌طور گسترده در این حوزه استفاده می‌شود. آشکارساز فیشر از هم‌دوسی بین سیگنال‌های دریافتی توسط حس‌گرهای مختلف در یک آرایه صوتی بهره می‌گیرد. در ادامه به معرفی مختصر این آشکارساز می‌پردازیم. فرض کنیم که یک آرایه متشکل از N حس‌گر، تغییرات فشار جوی را در بازه زمانی $t= 1,...,T$ اندازه‌گیری می‌کند، سیگنال‌های ثبت‌شده توسط حس‌گرها در یک ماتریس ذخیره می‌شوند، به‌گونه‌ای‌که هر سطر شامل سیگنال یک حس‌گر بوده و هر ستون، نمونه‌های زمانی سیگنال‌های حس‌گرها را در یک زمان مشخص نشان می‌دهد. در این صورت یک ماتریس $N\times T$ به‌دست می‌آید. ثبت‌های صورت‌گرفته در محور زمان پنجره‌گذاری می‌شوند و پردازش آرایه‌ای بر روی پنجره‌ها، به‌طور جداگانه صورت می‌گیرد؛ این بازه‌های زمانی بین[1] نامیده می‌شوند. متوسط هر ستون از این ماتریس به‌صورت زیر نمایش داده می‌شود [7]:

$$\bar{x}_t = \frac{1}{N}\sum_{n-1}^{N} x_{nt} \qquad (۱)$$

که در آن x_{nt} نمونه به‌صف‌شدهٔ متناظر با زمان t و حس‌گر nام است. داده‌های ثبت‌شده (حاوی نوفه) به‌صورت زیر مدل می‌شوند:

$$x_{nt} = \bar{x}_t + \epsilon_{nt} \qquad (۲)$$

که در آن ϵ_{nt} انحراف هر نمونه از مقدار متوسط است. نوفه در این مدل به‌صورت یک متغیر گوسی با متوسط صفر و واریانس σ^2 در نظر گرفته می‌شود. متوسط کل نمونه‌ها عبارت است از:

$$\bar{x} = \frac{1}{NT}\sum_{t=1}^{T}\sum_{n=1}^{N} x_{nt} \qquad (۳)$$

از آن‌جایی‌که انحراف هر ستون از متوسط کل را به‌صورت $\alpha_t = \bar{x}_t - \bar{x}$ می‌توان به‌دست آورد، لذا در این حالت مدل داده به‌صورت زیر خواهد بود:

$$x_{nt} = \bar{x} + \alpha_t + \epsilon_{nt} \qquad (۴)$$

با توجه به آنچه در بالا ذکر شد، تغییرات کل ثبت‌ها عبارت است از:

$$V_T = \sum_{t=1}^{T}\sum_{n=1}^{N}(x_{nt} - \bar{x})^2 \qquad (۵)$$

$$V_T = V_W + V_B, \qquad (۶)$$

$$V_W = \sum_{t=1}^{T}\sum_{n=1}^{N}(x_{nt} - \bar{x}_t)^2 \qquad (۷)$$

$$V_B = N\sum_{t=1}^{T}(\bar{x}_t - \bar{x})^2 \qquad (۸)$$

که در آن V_B معیاری از توان سیگنال هدف و V_w معیاری از توان نوفه در سیگنال ثبت شده است. با توجه به این که آشکارساز فیشر یک آشکارساز آماری است، توجه به توزیع آماری متغیرهای V_B و V_w بسیار مهم است. می‌توان نشان داد که امید ریاضی این متغیرها به‌صورت زیر است:

$$E(V_W) = T(N-1)\sigma^2 \qquad (۹)$$

$$E(V_B) = (T-1)\sigma^2 + N\sum_{t=1}^{T}\alpha_t^2 \qquad (۱۰)$$

که در آن σ^2 واریانس نوفه در هنگام ثبت سیگنال‌های حس‌گرهای مختلف است. با توجه به روابط بالا، $\frac{V_W}{T(N-1)}$ تخمینی بدون بایاس از واریانس است. این تخمین در شرایط H_0 و H_1 معتبر است. در مقابل عبارت $\frac{V_B}{T-1}$ تنها در شرایط H_0 که α_t صفر است؛ تخمینی بدون بایاس از واریانس است. با استفاده از دو رابطه بالا نسبت فیشر به‌صورت تقسیم دو واریانس تعریف می‌شود:

$$F = \frac{V_B/(T-1)}{V_W/T(N-1)} \qquad (۱۱)$$

روشن است درصورت صحیح‌بودن فرض H_0 مقدار خروجی آشکارساز، به‌طورتقریبی برابر یک و در غیر این صورت بیشتر از یک خواهد بود. با ساده‌سازی رابطه بالا آزمون فرض به‌صورت زیر در می‌آید:

$$F = \frac{T(N-1)}{N(T-1)}\frac{\sum_{t=1}^{T}(\sum_{n=1}^{N}x_{nt})^2 - \frac{1}{T}(\sum_{t=1}^{T}\sum_{n=1}^{N}x_{nt})^2}{\sum_{t=1}^{T}\sum_{n=1}^{N}x_{nt}{}^2 - \frac{1}{T}\sum_{t=1}^{T}(\sum_{n=1}^{N}x_{nt})^2} \qquad (۱۲)$$

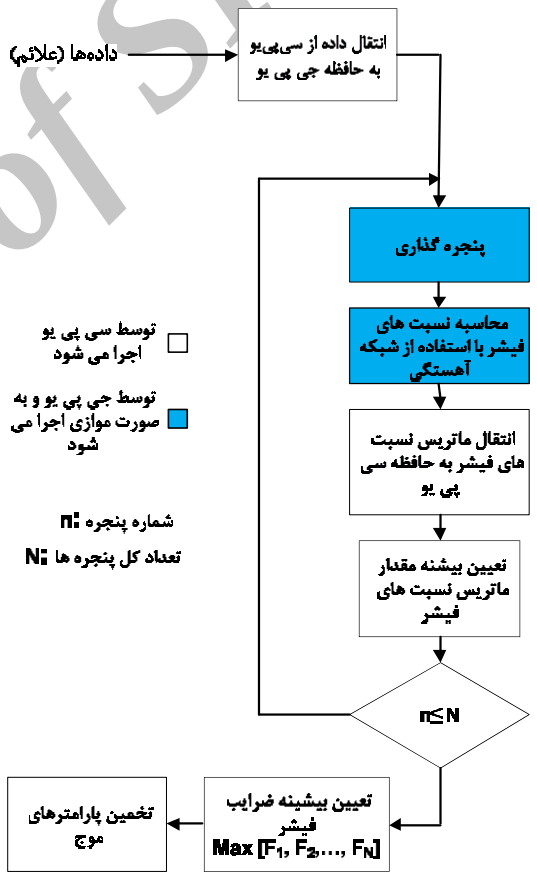
با توجه به نحوه تعریف V_B و V_w می‌توان نشان داد، توزیع احتمال آشکارساز فیشر در شرایط H_0، یک توزیع اف مرکزی با پارامترهای $v_1 = T-1$ و $v_2 = T(N-1)$ و

[1] Bin

در شرایط H_1، یک توزیع-اِف غیرمرکزی با پارامتر غیرمرکزیِ $\lambda_{nc} = v_1.snr$ است.

۳- روش پیشنهادی پیاده‌سازی الگوریتم فیشر

برای درک بهتر روش پیشنهادی پیاده‌سازی الگوریتم فیشر، روند اجرا در (شکل-۲(۲) نمایش داده شده است. همان‌طور که مشاهده می‌شود، الگوریتم با انتقال داده‌ها از سی‌پی‌یو به حافظه سراسری جی‌پی‌یو آغاز و پس از این انتقال مراحل بعدی در جی‌پی‌یو انجام می‌شود؛ در مرحله پنجره‌گذاری، مجموعه دادگان به پنجره‌های زمانی با طول معین تقسیم و در هر پنجره، نسبت فیشر به‌ازای تمامی بردارهای کندی محاسبه و پارامترهای سیگنال (سرعت ظاهری و زاویه سمت‌پشتی) برای بردار کُندی متناظر با بزرگ‌ترین نسبت فیشر، تعیین می‌شوند.



(شکل-۲): روندنما و مراحل پیاده‌سازی الگوریتم فیشر.
(Figure-2): Flowchart and implementation steps of Fisher algorithm.

لازم به توضیح است که در روش پیشنهادی، محاسبه نسبت فیشر به‌ازای بردارهای کُندی مختلف به‌صورت موازی انجام می‌شود. به‌دلیل آن که تعداد زیادی بردار کندی وجود دارد، انتظار می‌رود که با موازی‌سازی پردازش، زمان اجرای الگوریتم به‌طور قابل توجهی کاهش یابد. نسبت فیشر متناظر با هر بردار کندی، در یک ماتریس ذخیره می‌شود. پس از تکمیل این ماتریس توسط جی‌پی‌یو، نتایج حاصل دوباره به سی‌پی‌یو بازگردانده می‌شود. در ادامه، بیشینه این نسبت‌ها توسط سی‌پی‌یو محاسبه می‌شود و پس از آن که بیشینه نسبت فیشر به‌ازای تمامی پنجره‌ها محاسبه شد، به تعداد پنجره‌ها ضریب خواهیم داشت، $[F_1, F_2, \ldots, F_N]$. بیشینه این مقادیر مربوط به پنجره‌ای است که بیشترین توان سیگنال منبع صوت را در خود جای داده است و بیش‌ترین هم‌بستگی را بین داده‌های ثبت‌شده توسط حس‌گرهای آرایه دارد. بردار کندی متناظر با این نقطه، سرعت ظاهری و راستای منبع صوت را تخمین می‌زند.

مهم‌ترین پارامترهای مورد استفاده در پیاده‌سازی الگوریتم فیشر عبارتند از: آستانه نسبت فیشر، تفکیک‌پذیری نمونه‌برداری در شبکه بردار کندی، طول پنجره‌های زمانی، مقدار هم‌پوشانی پنجره‌ها و نرخ نمونه‌برداری. مقدار آستانه (ترشولد) آشکارساز فیشر توسط احتمال هشدار غلط[1] قابل تحمل در سامانه محاسبه می‌شود. تعیین آستانه در گام‌های اولیه نیاز به نظرات محققین خبره در این حوزه دارد تا با استفاده از آن و توزیع F[2] آشکارساز فیشر، آستانهٔ مناسب مشخص شود. تفکیک‌پذیری نمونه‌برداری از شبکه کندی به طور مستقیم بر تفکیک‌پذیری جستجوی سرعت و راستای موج تخمینی اثر می‌گذارد؛ از طرفی هرچه شبکه کندی با تفکیک‌پذیری بالاتری نمونه‌برداری شود، بار محاسباتی آشکارسازی افزایش می‌یابد. انتخاب طول پنجره‌های زمانی مناسب، نقش مؤثری در خروجی آشکارساز فیشر در هر پنجره دارد. اگر پنجره زمانی بسیار کوچک در نظر گرفته شود، به‌طوری‌که مقدار آن کمتر از زمان سیر موج از حس‌گر مرجع تا حس‌گر n باشد، در این‌صورت بین این دو حس‌گر در پنجره موجود، هم‌بستگی وجود نخواهد داشت و نمی‌توان از فزونی اطلاعات این حس‌گرها استفاده کرد؛ بنابراین درنظرگرفتن پنجره‌های زمانی بسیار کوچک به تشخیص یک بردار کندی اشتباه منجر خواهد شد. از سوی دیگر هرچه پنجره‌های زمانی بزرگ‌تر باشد، تغییرات مرحله در آن‌ها بیشتر است و هم‌بستگی بین سیگنال حس‌گرها کاهش

[1] False alarm (FA)
[2] F distribution

می‌یابد. در این حالت، بیشینه نسبت فیشر در پنجره‌ها کاهش می‌یابد. در بخش مربوط به شبیه‌سازی مقادیر پارامترهای تنظیم‌شده ارائه می‌شود.

(جدول-۱): پارامترهای سیگنال مرجع.
(Table-1): Reference signal parameters.

نام پارامتر	مقدار
بسآمد مرکزی	۳ هرتز
بسآمد نمونه‌برداری	۴۰ هرتز
فاکتور تضعیف	۰٫۳
هم‌پوشانی پنجره‌ها	۸۰٪
طول پنجره	۱۰۲۴ نمونه

۴- نتایج شبیه‌سازی‌ها و بحث

در این بخش پس از ارائه مدل سیگنال و چینش آرایه به بررسی نتایج پیاده‌سازی الگوریتم فیشر بر روی سی‌پی‌پو و جی‌پی‌یو می‌پردازیم. در انتها مقایسه سرعت اجرای این الگوریتم در دو معماری سی‌پی‌پو و جی‌پی‌یو ارائه می‌شود.

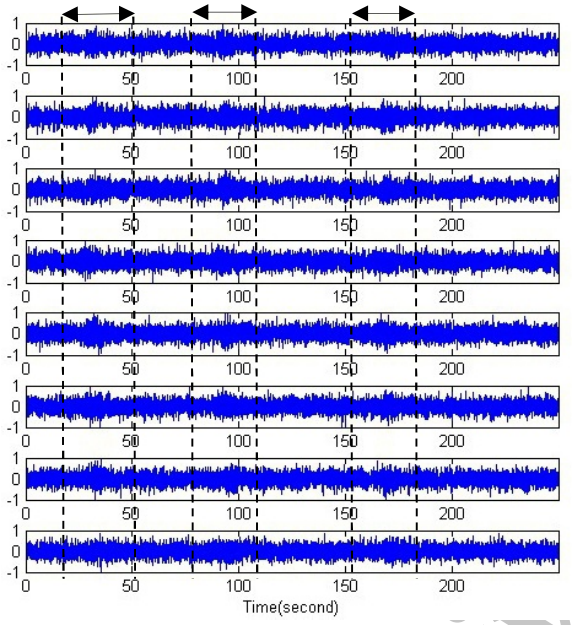
۴-۱- مدل سیگنال و چینش آرایه

مدل سیگنال استفاده‌شده در شبیه‌سازی‌های این بخش از رابطهٔ زیر پیروی می‌کند:

$$s(t) = ate^{-at}\sin(2\pi ft) \qquad (۱۳)$$

که از سه بخش سینوسی، نمایی و خطی تشکیل شده است. f بسامد مرکزی سیگنال و a فاکتور تضعیف نام دارد. نرخ تضعیف سیگنال مرجع، توسط پارامتر a قابل تنظیم است. پارامترهای سیگنال مرجعِ شبیه‌سازی‌شده در (**جدول-۱**(۱) نمایش داده شده است. به‌منظور شبیه‌سازی سیگنال خروجی هر حس‌گر، سه منبع صوت در راستاهای ۹۰، ۶۰ و ۱۲۰ درجه به‌ترتیب با سرعت‌های انتشار موج ۳۶۰، ۳۴۰ و ۳۹۰ متربرثانیه در نظر گرفته شده است. لازم به ذکر است که منظور از سرعت انتشار، سرعت ظاهری انتشار موج نسبت به آرایه است. با توجه به زوایا و سرعت‌های انتخاب‌شده و همین‌طور موقعیت مکانی حس‌گرها، سیگنال خروجی هر یک از حس‌گرها به‌ازای نسبت سیگنال به نوفه ۵- دسی‌بل به‌صورت شکل (۳) در می‌آید. موقعیت زمانی حضور سیگنال‌های منبع (به‌ازای سه راستای ذکرشده) به‌طور تقریبی توسط خط‌چین نمایش داد شده است.

آرایه صوتی در نظر گرفته‌شده در این مقاله، متشکل از ۸ حس‌گر است. پیکربندی آرایه در شکل (۴) مشاهده می‌شود. این چینش به‌دلیل وجود تقارن نسبی نسبت به راستاهای مختلف انتخاب شده است. پاسخ آرایه مربوط به این چینش به‌ازای بسآمدهای ۱ و ۲ هرتز در شکل (۵) نمایش داده شده است. دو دایره رسم‌شده در این شکل محدوده پاسخ آرایه برای بسامدهای ۱ و ۲ هرتز را نشان می‌دهند. توجه به این نکته ضروری است که طراحی چیدمان آرایه نقش مهمی در عملکرد الگوریتم دارد. این چیدمان باید بر اساس بازهٔ بسامدی پردازش و تفکیک‌پذیری زاویه‌ای مورد نیاز طراحی شود.

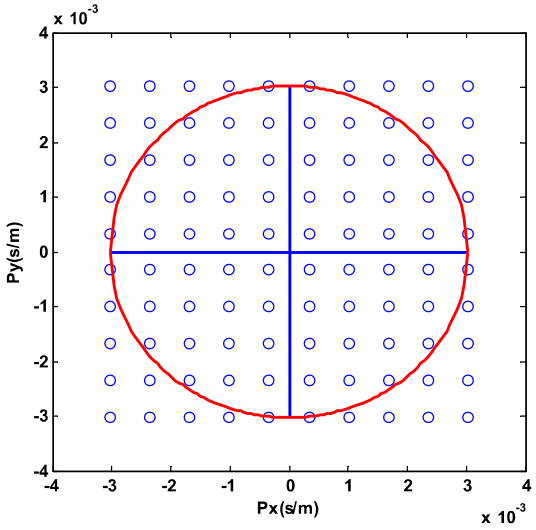


(شکل-۳): سیگنال شبیه‌سازی‌شده حس‌گرهای آرایه با سه مؤلفه از سه راستای ۹۰، ۶۰ و ۱۲۰ درجه. موقعیت زمانی حضور سیگنال منبع در هر حس‌گر به‌صورت تقریبی با خط‌چین نمایش داده شده است.

(Figure-3): Simulated signals of array sensors from three different directions of 90, 60, and 120 degrees. Approximate time-domain locations of source signals are presented by dotted lines.

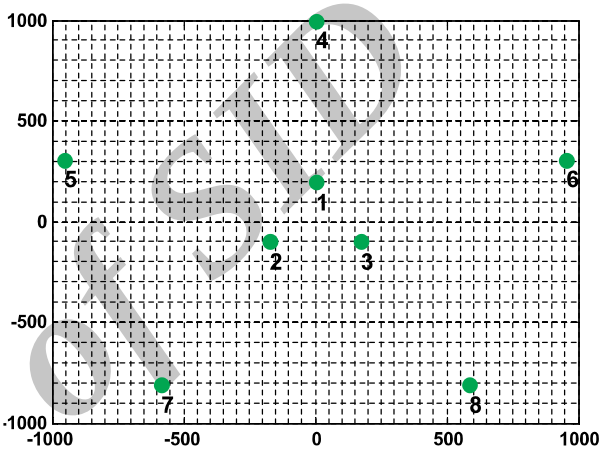
شبکه کُندی به‌کاربرده‌شده در این پژوهش در شکل (۶) مشاهده می‌شود. هر برداری کندی نماینده یک سرعت ظاهری و جهت موج دریافتی است. فرض می‌شود که موج ورودی از تمام جهات و با سرعت‌های ظاهری مختلف می‌تواند وارد شود. برای مقادیر پیش‌فرض برداری کُندی، فرض شده است که نمونه‌های شبکه به‌صورت یکنواخت انتخاب شده‌اند (یعنی فاصله بین هر دو سرعت متوالی، یکسان است). تعداد صد نمونه از هر سطر و ستون این شبکه دریافت شده و درمجموع ده‌هزار بردار کندی را تشکیل می‌دهند. البته به‌دلیل حصول وضوح بیشتر، تمام نمونه‌ها در شکل (۶) نمایش داده نشده‌اند. این عدد (۱۰۰×۱۰۰) برابر با

تعداد کل رشته‌های ایجادشده در کِرنل موازی جی‌پی‌یو است. دایره قرمز در شکل (۶) مجموعه بردارهای کُندی با سرعت ۳۳۰ متر بر ثانیه را نشان می‌دهد. با توجه به اینکه کمینه سرعت ظاهری موج ۳۳۰ متر بر ثانیه فرض می‌شود، جستجوی شبکه کُندی تنها شامل نقاط داخل این دایره می‌شود. با افزایش تعداد نمونه‌های بردار کندی در داخل دایره می‌توان دقت الگوریتم را افزایش داد. این مسئله زمان پردازش را افزایش می‌دهد. به‌عبارت دیگر، با به‌کارگیری پردازش موازی دقت الگوریتم را بهبود می‌توان داد و در عین حال، پردازش نیز به‌صورت بلادرنگ انجام شود.



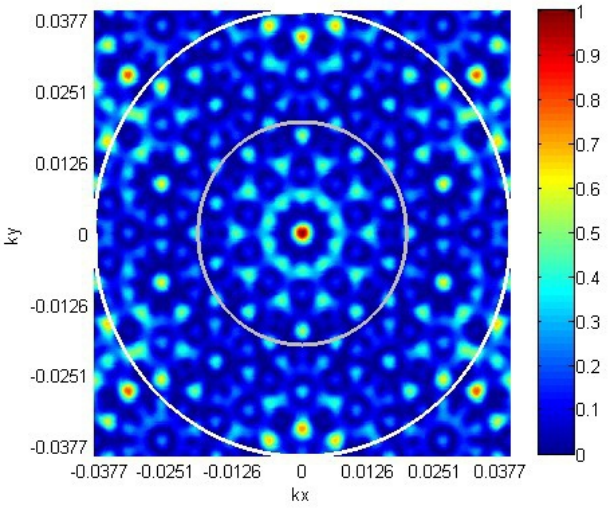
(شکل-۶): نمایش شبکه کندی. نقاط داخل دایره سرعت‌های بیش‌تر از ۳۳۰ متر بر ثانیه را مشخص می‌کند.

(Figure-6): Slowness-network illustration. The points inside the circle represent the source wave velocities higher than 330 m/s.



(شکل-۴): پیکربندی آرایه صوتی (فواصل بر حسب متر).

(Figure-4): Acoustic array configuration (distances are in meters).



(شکل-۵): پاسخ آرایه شکل (۴) به‌ازای بسامدهای ۱ و ۲ هرتز.

(Figure-5): Array response for the frequencies of 1 and 2 Hz.

پردازندهٔ مورد استفاده در این پژوهش، یک سی‌پی‌یو چهار هسته‌ای i5 شرکت اینتل با بسامد ساعت ۳ گیگاهرتز است. از آن‌جایی که سرعت اجرای الگوریتم به قابلیت محاسباتی جی‌پی‌یو مورد استفاده نیز بستگی دارد، زمان اجرای الگوریتم با استفاده از دو پردازنده گرافیکی متفاوت یکی با مدل Geforce GT 630، قابلیت محاسباتی ۲٫۱ و تعداد کل هسته ۹۶ (جی‌پی‌یو شماره ۱) و دیگری Geforce GTX 650 TI Boost با قابلیت محاسباتی ۳٫۰ و تعداد کل هسته ۷۶۸ (جی‌پی‌یو شماره ۲) به‌دست آمده و مقایسه شده است. در شبیه‌سازی‌های این قسمت از نرم‌افزار Visual Studio 2010 و CUDA toolkit 5.0 استفاده شده است.

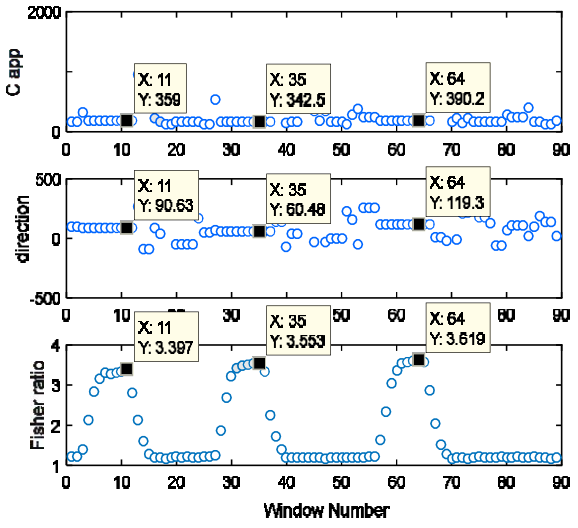
در ادامه نتایج پیاده‌سازی آشکارساز فیشر بر روی سی‌پی‌یو و جی‌پی‌یو تشریح می‌شود.

۲-۴- نتایج تخمین پارامتر موج

پس از شبیه‌سازی سیگنال‌های دریافتی در محل حس‌گرها، الگوریتم یک بار بدون اعمال پردازش موازی و بار دیگر با اعمال پردازش موازی با استفاده از جی‌پی‌یو پیاده‌سازی شد. در این بخش نتایج پیاده‌سازی الگوریتم فیشر بر روی سی‌پی‌یو و جی‌پی‌یو ارائه می‌شود. در شکل (۷) و شکل (۸) خروجی آشکارساز فیشر به‌ترتیب بر روی سی‌پی‌یو و جی‌پی‌یو نمایش داده شده است. هر یک از این شکل‌ها سه

نمودار دارد. منحنی پایین، خروجی آماره آزمون[1] آشکارساز فیشر را به‌ازای پنجره‌های زمانی مختلف نمایش می‌دهد. پنجره‌های متوالی دارای هم‌پوشانی ۸۰٪ هستند. منحنی وسط و بالا به‌ترتیب راستا و سرعت تخمین زده‌شده در هر پنجرهٔ زمانی را نشان می‌دهد.

همان‌طور که در منحنی‌های مربوط به نسبت فیشر (منحنی پایین) در شکل (۷) و شکل (۸) مشاهده می‌شود، الگوریتم فیشر حضور سه سیگنال را در پنجره‌های زمانی با شماره‌های تقریبی ۱۱، ۳۵ (و ۳۳ در سی‌پی‌یو) و ۶۴ آشکارسازی کرده است. مقادیر راستا و سرعت تخمینی متناظر با هر یک از این سه پنجره در جدول (۲) ارائه شده است. همانطور که مشاهده می‌شود، پارامترهای تخمین‌زده‌شده توسط جی‌پی‌یو با دقت قابل قبولی، تخمین‌های محاسبه‌شده توسط سی‌پی‌یو را دنبال می‌کنند که پیاده‌سازی صحیح کد را نشان می‌دهد. شایان ذکر است که تفاوت اندک میان مقادیر تخمینی توسط سی‌پی‌یو و جی‌پی‌یو، ناشی از مواردی هم‌چون دقت متفاوت اعداد با ممیز-شناور[2] در محاسبات انجام‌شده توسط سی‌پی‌یو و جی‌پی‌یو و هم‌چنین نوفه کانال تصادفی تولیدی در شبیه‌سازی‌ها است.

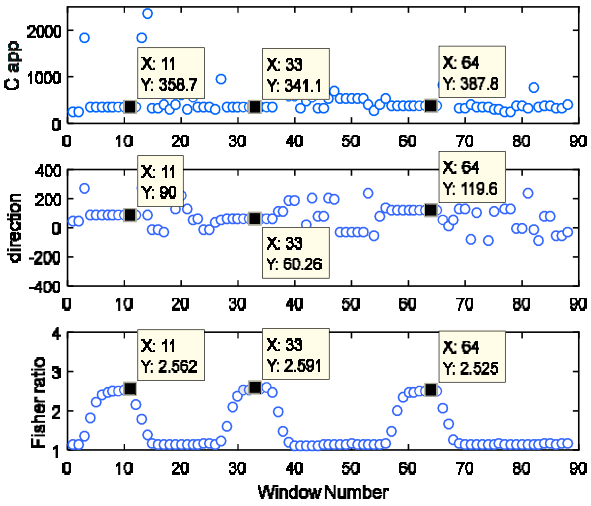


(شکل-۷): نتیجه آشکارساز فیشر با استفاده از سی‌پی‌یو به‌ازای سیگنالی با سه مؤلفه از سه راستای ۹۰، ۶۰ و۱۲۰درجه و سرعت‌های ۳۶۰،۳۴۰ و ۳۹۰ متر بر ثانیه.

(Figure-7): The results of Fisher detector implementation using CPU for a signal received from three sources with directions 90, 60, and 120 with velocities 340, 360, and 390 m/s, respectively.

[1] Test Statistic

[2] Floating-point



(شکل- ۸): نتیجه آشکارساز فیشر با استفاده از جی‌پی‌یو شماره ۱ به‌ازای سیگنالی با سه مؤلفه از سه راستای ۹۰، ۶۰ و۱۲۰ درجه و سرعت‌های ۳۶۰،۳۴۰ و ۳۹۰ متر بر ثانیه.

(Figure-8): The results of Fisher detector implementation using the GPU no.1 for a signal received from three sources with directions 90, 60, and 120 with velocities 340, 360, and 390 m/s, respectively.

(جدول-۲) مقایسه پارامترهای تخمین زده‌شده توسط سی‌پی‌یو و جی‌پی‌یو.

(Table-2): Comparison between the estimated parameters from CPU and GPU implementations.

پارامتر	سی‌پی‌یو	جی‌پی‌یو
راستای تخمینی (درجه)	119.30	119.6
	60.48	60.26
	90.63	90.00
سرعت تخمینی (متر بر ثانیه)	390.2	387.8
	342.5	341.1
	359	358.7

۳-۴- مقایسه سرعت پیاده‌سازی بر روی سی‌پی‌یو و جی‌پی‌یو

در این بخش به مقایسه سرعت پیاده‌سازی الگوریتم فیشر بر روی سی‌پی‌یو و جی‌پی‌یو می‌پردازیم. جدول (۳) نتایج مقایسه زمان لازم برای اجرای الگوریتم به‌ازای شبکه‌های کندی با ابعاد ۵۰×۵۰، ۱۰۰×۱۰۰، ۱۵۰×۱۵۰، ۲۰۰×۲۰۰ و ۲۵۰×۲۵۰ را نشان می‌دهد. همان‌گونه که انتظار می‌رود با افزایش تعداد نمونه‌های شبکه کندی حجم بیش‌تری از محاسبات به‌صورت موازی انجام شده و درنتیجه نسبت بهبود سرعت جی‌پی‌یو به سی‌پی‌یو افزایش می‌یابد. در این جدول نتایج به‌ازای هر دو جی‌پی‌یو ذکرشده در قبل ارائه شده است. با توجه به این که ظرفیت اجرای موازی

محاسبات در سخت‌افزارهای جی‌پی‌یو محدود است، می‌توان مشاهده کرد که با افزایش بیش‌تر ابعاد شبکه کندی نرخ افزایش سرعت کاهش می‌یابد. در جدول (۴) تأثیر طول پنجره در افزایش سرعت پیاده‌سازی بررسی شده است. در این جدول زمان اجرا به‌ازای پنجره با طول‌های ۲۵۶، ۵۱۲ و ۱۰۲۴ نمونه ارائه شده است. همان‌طور که مشاهده می‌شود، افزایش طول پنجره تغییر محسوسی در سرعت اجرای الگوریتم توسط جی‌پی‌یو ندارد؛ دلیل این امر این است که اگرچه با کاهش طول پنجره، تعداد تکرار حلقه موجود در شکل (۲) افزایش می‌یابد؛ ولی محاسبات مورد نیاز برای پردازش داده‌های هر پنجره به همان نسبت کاهش می‌یابد؛ بنابراین با تغیر طول پنجره سرعت پیاده‌سازی الگوریتم توسط جی‌پی‌یو به‌طورتقریبی ثابت باقی می‌ماند. این در حالی است که با کاهش طول پنجره تعداد تکرار جستجوی شبکه کُندی افزایش یافته و زمان اجرای برنامه توسط سی‌پی‌یو بیشتر می‌شود.

(جدول-۳): نتایج مقایسه زمان لازم برای پیاده‌سازی الگوریتم فیشر بر روی سی‌پی‌یو و جی‌پی‌یو به‌ازای طول پنجره ۱۰۲۴ نمونه و هم‌پوشانی ۸۰٪ (اعداد برحسب میلی‌ثانیه است).

(Table-3): Results of time-consumptions measurement for the CPU and GPU implementations of Fisher detector with window length of 1024 samples and overlap rate of 80% (times are in milliseconds).

ابعاد شبکه کُندی	زمان سی‌پی‌یو	زمان جی‌پی‌یو شماره ۱	نسبت افزایش سرعت ۱	زمان جی‌پی‌یو شماره ۲	نسبت افزایش سرعت ۲
50×50	18.5	3.9	4.8	1.0	18.7
100×100	74.1	8.5	8.7	2.2	33.7
150×150	164.9	16.2	10.3	4.2	39.8
200×200	299.2	27.7	10.7	6.5	45.8
250×250	460.3	42.4	10.9	9.5	48.7

(جدول-۴) نتایج زمانی پیاده‌سازی الگوریتم فیشر روی سی‌پی‌یو و جی‌پی‌یو به‌ازای پنجره با طول‌های مختلف (ابعاد شبکه کندی ۱۰۰× ۱۰۰ می‌باشد).

(Table-4): Results of time-consumptions measurement for the CPU and GPU implementations of Fisher detector with different window lengths (for slowness network size of 100×100).

طول پنجره	زمان سی‌پی‌یو	زمان جی‌پی‌یو شماره ۱	نسبت افزایش سرعت ۱	زمان جی‌پی‌یو شماره ۲	نسبت افزایش سرعت ۲
L = 256	139.4	9.5	14.7	2.8	49.8
L = 512	102.5	8.5	12.1	2.5	41
L = 1024	74.1	8.6	8.6	2.2	33.7

۵- نتیجه‌گیری

آشکارساز فیشر یکی از مهم‌ترین ابزارهای پردازشی سیگنال‌های بسامد پایین در حوزهٔ تخمین پارامترهای موج (راستا و سرعت) است. در این مقاله با استفاده از پردازش موازی مبتنی بر جی‌پی‌یو به پیاده‌سازی آشکارساز فیشر پرداختیم. نتایج شبیه‌سازی‌ها نشان می‌دهد که با استفاده از جی‌پی‌یو سرعت اجرای این الگوریتم را تا حدود پنجاه برابر نسبت به سی‌پی‌یو می‌توان افزایش داد. البته این بهبود با استفاده از جی‌پی‌یوهای با قدرت محاسبات بالاتر قابل ارتقا است. نتایج بیان‌گر آن است که پردازش موازی مبتنی بر جی‌پی‌یو امکان تخمین پارامترهای موج و آشکارسازی سیگنال را (با استفاده از الگوریتم‌های موجود در این حوزه) به‌صورت بلادرنگ تسهیل می‌کند.

6-References ۶- مراجع

[1] V. L. Zimmer, N. Sitar, "Detection and location of rock falls using seismic and infrasound sensors," *Engineering Geology*, vol. 193, pp. 49-60, Apr. 2015.

[2] R.D. Costley, W. G. Frazier, et al., "Frequency-wavenumber processing for infrasound distributed arrays," *Journal of Acoustical Society of America*, vol. 134, no. 4, pp.EL307-EL311, 2013.

[3] S. Havens, H.-P. Marshall, *et al.*, *Real-Time Avalanche Detection for High Risk Areas*, Research Report, Transportation Department, Idaho University, Dec. 2014, available online at: https://itd.idaho.gov/highways/research/archived/reports/RP219Final12312014.pdf.

[4] W. W. Arrasmith, E. R. Coots, J. V. Olson, and E. A. Skowbo, "Analyzing infrasound and seismic signals emanating from a waterborne system using canonical modeling and analysis methods", International Journal of Modeling and Optimization, vol. 4, no. 3, pp. 176-181, Jun. 2014.

[5] M. Charbit, I Che, and A Le Pichon, "Asymptotic distribution of GLRT versus Fisher distribution for infrasonic detection," *Geophysical Research Abstracts*, vol. 15, EGU2013-3690, 2013.

[6] J. Park, B. W. Stump, C. Hayward, Detection of regional infrasound signals using array data: testing, tuning and physical interpretation, Journal of Acoustical Society of America, vol. 140, no. 1, pp. 240-259, Jul. 2016.

[7] S. J. Arrowsmith, R. Whitaker, S. R. Taylor, "Regional monitoring of infrasound events using multiple arrays: application to Utah and Washington State", *Geophysical Journal International*, pp. 291–300, Jul. 2008.

[8] S. J. Arrowsmith, R. Whitaker, C. Katz, C. Hayward, "The *F*-detector revisited: An improved strategy for signal detection at seismic and infrasound arrays", *Seismological Society of America*, vol. 99, no. 1, pp. 449–453, 2009.

[9] B. Melton, and L. Baily, "Multiple signal correlators," *Geophysics, XXII* (3), pp. 565-588, 1957.

[10] R. R. Blandford, "An automatic event detector at the Tonto Forest seismic observatory", *Geophysics*, vol. 39, pp. 633, 1974.

[11] S. Angelis *et al.*, "Detecting hidden volcanic explosions from Mt. Cleveland Volcano, Alaska with infrasound and ground-coupled airwaves," *Geophysical Research Letters*, vol. 39, 2012.

[12] L. G. Evers and H. W. Haak, "Tracing a meteoric trajectory with infrasound", *Geophysical Research Letters*, vol. 30, no. 24, Dec. 2003.

[13] Y. Cansi, "An automatic seismic event processing for detection and location: the PMCC method", *Geophysical Research Letters*, vol. 22, pp. 1021-1024,1995.

[14] J. Nickolls, W.J. Dally, "The GPU computing era," *IEEE Micro Magazine*, vol. 30, pp. 56-69, 2010.

[15] H. Chen, S. Saïghi, L. Buhry, and S. Renaud, "Real-time simulation of biologically realistic stochastic neurons in VLSI," *IEEE Transactions on Neural Networks*, vol. 21, no. 9, pp. 1511–1517, Sep. 2010.

[16] S. U. Gjerald, R. Brekken, T. Hergum, J. D'hoog "Real-time ultrasound simulation using the GPU," *IEEE Transactions on Ultrasonic Ferroelectrics and Frequency Control*, vol. 59, pp. 885-892, 2012.

[17] Y. Dai, *et al.*, "Real-time visualized freehand 3D ultrasound reconstruction based on GPU," *IEEE Transactions on Information Technology in Biomedicine*, vol. 14, pp. 1338-1345, 2010.

[18] C. Richter, S. Schops, and M. Clemens, "GPU acceleration of finite difference schemes used in coupled electromagnetic/ thermal field simulations", *IEEE Transactions on Magnetics*, vol. 49, no. 5, pp. 1649-1652, May 2013.

[19] W. Rodrigues, *et al.*, "Accelerating atomistic calculation of quantum energy eigenstates on graphic cards", *Computer physics Communications*, pp. 2510-2518, May 2014.

[20] A. Artu, "Parallel wavelet-based clustering algorithm on GPUs using CUDA", *Procedia Computer Science*, vol. 3, pp. 396-400, 2011.

[21] L. Mussi, F. Daolio, S. Cagnoni, "Evaluation of parallel particle swarm optimization algorithms within the CUDA™ architecture", *Information Sciences*, pp. 4642-4657, 2011.

[22] D. B. Kirk, W.H. Wen-Mei, *Programming Massively Parallel Processors: A Hands-on Approach*, 2nd Edition, Morgan Kaufmann, 2011.

حامد صادقی مدرک کارشناسی خود را در رشته الکترونیک در سال ۱۳۸۴ از دانشگاه شاهد اخذ کرد؛ سپس دوره‌های کارشناسی ارشد و دکترای خود را در دانشگاه تربیت مدرس از سال ۱۳۸۷ تا ۱۳۹۲ به اتمام رساند. ایشان در زمینه‌های مخابرات بی‌سیم، پردازش سیگنال‌های صوتی و تئوری آشکارسازی سیگنال، پژوهش‌های خود را پیش می‌برد.

نشانی رایانامه ایشان عبارت است از:

h.sadeghi.2015@ieee.org

امیر اخوان مدرک کارشناسی مخابرات خود را در سال ۱۳۸۷ از دانشگاه صنعتی اصفهان، کارشناسی ارشد خود را در سال ۱۳۹۰ از دانشگاه تربیت مدرس و دکترای مهندسی پزشکی خود را در سال ۱۳۹۶ از دانشگاه صنعتی امیرکبیر دریافت کرد. ایشان هم‌اکنون استادیار دانشگاه صنعتی همدان بوده و پژوهش‌های خود را در زمینه‌های پردازش سیگنال‌های صوتی و سیگنال‌های حیاتی به‌پیش می‌برد.

نشانی رایانامه ایشان عبارت است از:

amir.akhavan@aut.ac.ir